



How I Use Laravel Queues

Accept the work safely.
Let a worker do it.
Know whether it finished.

REQUEST

Accept and enqueue



WORKER

Execute and recover



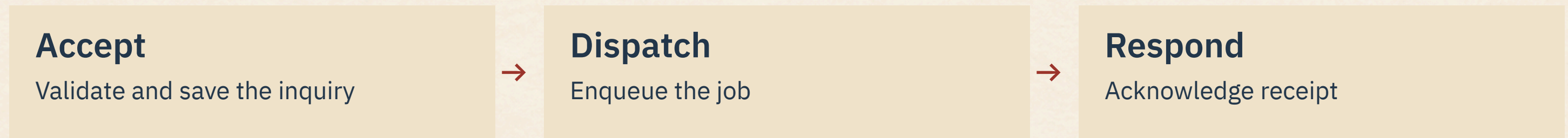
RESULT

Make completion visible

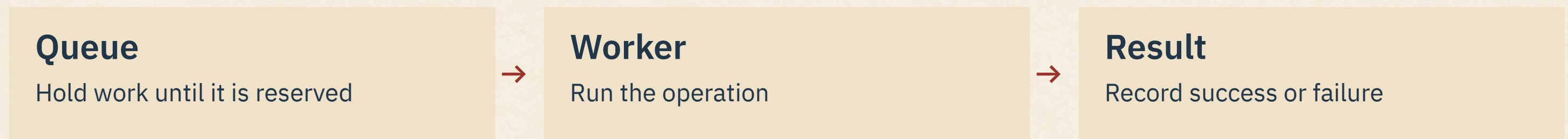
The request hands off. The worker executes.

A response can succeed while the queued work is still waiting or has failed.

HTTP REQUEST



SEPARATE WORKER



A worker may start before the HTTP response. Enqueueing is not proof of completion.

A Job class is only part of the setup.

Check how the job is dispatched, where it lands, and what consumes it.

01 ShouldQueue

The regular dispatcher uses this contract to decide whether to queue the job.

02 Connection and queue

sync runs in the current process. database or redis can use a separate worker.

03 Worker lifecycle

Listen to the correct connection and queue. Restart long-running workers for new code.

A worker listening only to default will not consume a job waiting in emails.

After commit is not after the response.

Choose when to enqueue based on the data and backend the job depends on.

01 Redis before commit

A worker may start before a newly created record is visible to its database connection.

02 Dispatch after commit

`afterCommit()` or `after_commit` can hold enqueueing until open transactions commit.

03 This site's database queue

The inquiry and job insert share one database transaction. They commit or roll back together.

Here, `beforeCommit()` keeps the job insert inside that shared database transaction.

A retry should continue unfinished work.

Partial success is the case that exposes duplicate side effects.

-
- | | | |
|----|------------------------------------|--|
| 01 | Track each delivery | Record the owner notification and sender confirmation separately after successful sends. |
| 02 | Expect an uncertain result | The provider can accept an email before the application records its success. |
| 03 | Use stable idempotency keys | Keep the same key for the same operation. Respect the provider's supported window. |
-

This reduces duplicate-delivery risk. It is not an exactly-once email guarantee.

Retries need limits and a useful signal.

Choose recovery settings for the operation, rather than copying numbers blindly.

01 Attempts and backoff

This contact job uses three attempts with increasing delays between failures.

02 Timeout below retry interval

Its 60-second timeout is below the database queue's 90-second `retry_after` setting.

03 Notice incomplete work

Inspect failures, retry within the allowed window, and send older cases to manual review.

Configure external-call timeouts too. Log useful identifiers, not private message payloads.

Test the handoff and the operation separately.

A dispatch assertion cannot prove the worker's business operation succeeds.

01 Request boundary

Check accepted input queues the right work.
Check rejected input and atomic enqueue failure.

02 Job behavior

Use isolated data and external-service fakes.
Include partial failure and another attempt.

03 Operating environment

Check worker process, connection, queue,
and whether failures are visible.

A successful HTTP redirect alone does not prove a separate worker is healthy.

Can you follow one job all the way through?

If the user needs the result immediately, keep that expectation explicit in the interface.

- ☐ **Connection + queue** I know where the job lands.
- ☐ **Worker** A running worker consumes that connection and queue.
- ☐ **Safe retries** Another attempt does not blindly repeat completed side effects.
- ☐ **Completion + failure** I can tell whether the work finished and find failures.

[Read the full article and transaction example: How I Use Laravel Queues](#)