Job Expiration

In your ***config/queue.php*** configuration file, each queue connection defines a ***retry_after option***. This option specifies how many seconds the queue connection should wait before retrying a job that is being processed. For example, if the value of ***retry_after*** is set to 90, the job will be released back onto the queue if it has been processing for 90 seconds without being released or deleted. Typically, you should set the ***retry_after*** value to the maximum number of seconds your jobs should reasonably take to complete processing.

****

Note: The only queue connection which does not contain a retry_after value is Amazon SQS. SQS will retry the job based on the Default Visibility Timeout which is managed within the AWS console.

****

Worker Timeouts

The ***queue:work*** Artisan command exposes a ***--timeout option***. By default, the ***--timeout value*** is 60 seconds. If a job is processing for longer than the number of seconds specified by the timeout value, the worker processing the job will exit with an error. Typically, the worker will be restarted automatically by a process manager configured on your server:

php artisan queue:work --timeout=60

The retry_after configuration option and the ***--timeout CLI option*** are different, but work together to ensure that jobs are not lost and that jobs are only successfully processed once.

****

Note (Very Important): The ***--timeout value*** should always be at least several seconds shorter than your ***retry_after configuration*** value. This will ensure that a worker processing a frozen job is always terminated before the job is retried. If your ***--timeout option*** is longer than your ***retry_after configuration value***, your jobs may be processed twice.

****

Supervisor Configuration

In production, you need a way to keep your ***queue:work processes*** running. A ***queue:work process*** may stop running for a variety of reasons, such as an exceeded worker timeout or the execution of the ***queue:restart command***.

For this reason, you need to configure a process monitor that can detect when your ***queue:work processes*** exit and automatically restart them. In addition, process monitors can allow you to specify how many ***queue:work processes*** you would like to run concurrently. Supervisor is a process monitor commonly used in Linux environments and we will discuss how to configure it in the following documentation.

****

Dealing With Failed Jobs

Sometimes your queued jobs will fail. Don't worry, things don't always go as planned! Laravel includes a convenient way to specify the maximum number of times a job should be attempted. After an asynchronous job has exceeded this number of attempts, it will be inserted into the ***failed_jobs database table***. Synchronously dispatched jobs that fail are not stored in this table and their exceptions are immediately handled by the application.

A migration to create the ***failed_jobs table*** is typically already present in new Laravel applications. However, if your application does not contain a migration for this table, you may use the ***queue:failed-table command*** to create the migration:

php artisan queue:failed-table

php artisan migrate

****

When running a queue worker process, you may specify the maximum number of times a job should be attempted using the --tries switch on the queue:work command. If you do not specify a value for the --tries option, jobs will only be attempted once or as many times as specified by the job class' $tries property:

php artisan queue:work redis --tries=3

****

Using the ***--backoff option***, you may specify how many seconds Laravel should wait before retrying a job that has encountered an exception. By default, a job is immediately released back onto the queue so that it may be attempted again:

php artisan queue:work redis --tries=3 --backoff=3

****

If you would like to configure how many seconds Laravel should wait before retrying a job that has encountered an exception on a **per-job basis**, you may do so by defining a backoff property on your job class:

```
/**
 * The number of seconds to wait before retrying the job.
 *
 * @var int
 */
public $backoff = 3;
```

****

If you require more complex logic for determining the job's backoff time, you may define a backoff method on your job class:

```
/**
* Calculate the number of seconds to wait before retrying the job.
*/
public function backoff(): int
{
    return 3;
}
```

****

You may easily configure "exponential" backoffs by returning an array of backoff values from the backoff method. In this example, the retry delay will be 1 second for the first retry, 5 seconds for the second retry, and 10 seconds for the third retry:

```
/**
* Calculate the number of seconds to wait before retrying the
* job.
* @return array<int, int>
*/
public function backoff(): array
{
    return [1, 5, 10];
}
```

****

Cleaning Up After Failed Jobs

When a particular job fails, you may want to send an alert to your users or revert any actions that were partially completed by the job. To accomplish this, you may define a **failed method** on your job class. The **Throwable instance** that caused the job to fail will be passed to the **failed method**:

```
/**
* Handle a job failure.
*/
public function failed(Throwable $exception): void
{
    // Send user notification of failure, etc...
}
```

Note: A new instance of the job is instantiated before invoking the **failed method**; therefore, any class property modifications that may have occurred within the **handle method** will be **lost**.

****

Retrying Failed Jobs

To view all of the failed jobs that have been inserted into your **failed_jobs** database table, you may use the **queue:failed** Artisan command:

php artisan queue:failed

The **queue:failed** command will list the job ID, connection, queue, failure time, and other information about the job. The job ID may be used to retry the failed job. For instance, to retry a

failed job that has an ID of **ce7bb17c-cdd8-41f0-a8ec-7b4fef4e5ece**, issue the following command:

php artisan queue:retry ce7bb17c-cdd8-41f0-a8ec-7b4fef4e5ece

If necessary, you may pass multiple IDs to the command:

php artisan queue:retry ce7bb17c-cdd8-41f0-a8ec-7b4fef4e5ece 91401d2c-0784-4f43-824c-34f94a33c24d

You may also retry all of the failed jobs for a particular queue:

php artisan queue:retry --queue=name

To retry all of your failed jobs, execute the **queue:retry command** and pass all as the ID:

php artisan queue:retry all

If you would like to delete a failed job, you may use the **queue:forget** command:

php artisan queue:forget 91401d2c-0784-4f43-824c-34f94a33c24d

When using Horizon, you should use the **horizon:forget** command to delete a failed job instead of the **queue:forget** command.

To delete all of your failed jobs from the failed_jobs table, you may use the **queue:flush** command:

php artisan queue:flush

****

Ignoring Missing Models

When injecting an Eloquent model into a job, the model is automatically serialized before being placed on the queue and re-retrieved from the database when the job is processed. However, if the model has been deleted while the job was waiting to be processed by a worker, your job may fail with a **ModelNotFoundException**.

For convenience, you may choose to automatically delete jobs with missing models by setting your job's **deleteWhenMissingModels** property to true. When this property is set to true, Laravel will quietly discard the job without raising an exception:

```
/**
 * Delete the job if its models no longer exist.
 *
 * @var bool
 */
public $deleteWhenMissingModels = true;
```

****

Pruning Failed Jobs

You may prune the records in your application's **failed_jobs table** by invoking the **queue:prune-failed** Artisan command:

php artisan queue:prune-failed

By default, all the failed job records that are more than 24 hours old will be pruned. If you provide the **--hours option** to the command, only the failed job records that were inserted within the last N number of hours will be retained. For example, the following command will delete all the failed job records that were inserted more than **48 hours ago**:

php artisan queue:prune-failed --hours=48

****

Disabling Failed Job Storage

You may instruct Laravel to discard failed jobs without storing them by setting the **queue.failed.driver** configuration option's value to null. Typically, this may be accomplished via the **QUEUE_FAILED_DRIVER** environment variable:

QUEUE_FAILED_DRIVER=null

****

Failed Job Events

If you would like to register an event listener that will be invoked when a job fails, you may use the Queue facade's failing method. For example, we may attach a closure to this event from the boot method of the **AppServiceProvider** that is included with Laravel:

```
use Illuminate\Support\Facades\Queue;

use Illuminate\Support\ServiceProvider;

use Illuminate\Queue\Events\JobFailed;


class AppServiceProvider extends ServiceProvider

{

    /**

     * Register any application services.

     */

    public function register(): void

    {

        // ...

    }


    /**

     * Bootstrap any application services.

     */

    public function boot(): void
```

```
    {
        Queue::failing(function (JobFailed $event) {
            // $event->connectionName
            // $event->job
            // $event->exception
        });
    }
}
```

****

Clearing Jobs From Queues

When using Horizon, you should use the **horizon:clear** command to clear jobs from the queue instead of the **queue:clear** command.

If you would like to delete all jobs from the default queue of the default connection, you may do so using the **queue:clear** Artisan command:

php artisan queue:clear

You may also provide the connection argument and queue option to delete jobs from a specific connection and queue:

php artisan queue:clear redis --queue=emails

Note: Clearing jobs from queues is only available for the **SQS**, Redis, and database queue drivers. In addition, the **SQS** message deletion process takes up to **60 seconds**, so jobs sent to the SQS queue up to 60 seconds after you clear the **queue might also be deleted**.

****

Monitoring Your Queues

If your queue receives a sudden influx of jobs, it could become overwhelmed, leading to a long wait time for jobs to complete. If you wish, Laravel can alert you when your queue job count exceeds a specified threshold.

To get started, you should schedule the **queue:monitor** command to run every minute. The command accepts the names of the queues you wish to monitor as well as your desired job count threshold:

php artisan queue:monitor redis:default,redis:deployments --max=100

Scheduling this command alone is not enough to trigger a notification alerting you of the queue's overwhelmed status. When the command encounters a queue that has a job count exceeding your threshold, an **Illuminate\Queue\Events\QueueBusy** event will be dispatched. You may listen for this event within your application's **EventServiceProvider** in order to send a notification to you or your development team:

```
use App\Notifications\QueueHasLongWaitTime;
use Illuminate\Queue\Events\QueueBusy;
use Illuminate\Support\Facades\Event;
use Illuminate\Support\Facades\Notification;

/**
 * Register any other events for your application.
 */
public function boot(): void
{
    Event::listen(function (QueueBusy $event) {
        Notification::route('mail', 'dev@example.com')
                ->notify(new QueueHasLongWaitTime(
                    $event->connection,
                    $event->queue,
                    $event->size
                ));
    });
}
```

****