Documentation: Porting Laravel Queues to Cloudflare Queues

□ Overview

Porting Laravel's native (or existing) queue system to **Cloudflare Queues** is a **non-trivial task**, because it involves bridging two different runtime and infrastructure models:

- **Laravel** — PHP-based, long-running workers, internal job serialization, etc.
- **Cloudflare Queues** — Distributed queue system built for Cloudflare Workers / HTTP API model.

You must consider:

- Job serialization & dispatching
- Message consumption & acknowledgment
- Retries & error handling
- Environment & runtime differences

This documentation outlines:

- The key differences between Laravel and Cloudflare Queues
- A strategy to port Laravel Queues
- Example configurations and approaches
- Common challenges and recommendations

⚙ Key Differences Between Laravel Queues and Cloudflare Queues

Feature	Laravel (Redis / Database / SQS)	Cloudflare Queues
Producer Model	Laravel dispatches jobs (serialized PHP classes) to the queue driver (Redis, SQS, etc.)	Any code (Worker or HTTP API) can push JSON messages via Cloudflare Queue API
Consumer / Worker	`php artisan queue:work` or Horizon continuously polls and processes messages	Cloudflare Worker's `queue` handler or HTTP pull consumer fetches batches (Cloudflare Docs)
Acknowledgement / Retries	Removes job on success; failed jobs retried or moved to failed jobs table	Supports message acknowledgment, retries, dead-letter queue, batching, delays
Serialization / Payload	Serializes PHP job class and data	Messages are JSON — cannot run PHP inside Workers; need to transform or process externally
Environment / Runtime	PHP + Laravel with all dependencies	Cloudflare Worker (JavaScript/TypeScript) or external consumer — limited runtime, fast execution

---

□ Porting Strategy (Step-by-Step)

1. Decide Your Consumer Environment

You have two main options:

Option A — Run Consumer Inside Cloudflare Worker

- Use the Cloudflare `queue` handler to process messages directly inside a Worker.
- Suitable for **lightweight tasks** (e.g. sending emails, webhooks).
- Offers **low latency** and **global scalability**.
- Docs: Cloudflare Queues - Getting Started

Option B — Run Consumer in Laravel (Pull Consumer)

- Laravel fetches messages from Cloudflare Queue using the API.
- Jobs are processed in the Laravel environment (PHP).
- Easier for jobs that need Laravel's features (database, mail, logging).

✅ For beginners, **Option B** is simpler and allows reuse of existing Laravel job classes.

---

2. Add a Custom "Bridge" Queue Driver in Laravel

You need Laravel to send jobs into Cloudflare Queues instead of Redis/SQS.

Steps:

1. Create (or install) a Laravel driver that connects to Cloudflare Queues.
   Example package: **ayles-software/laravel-cloudflare-queue**
2. Add the configuration in `config/queue.php`:

```
'connections' => [
    'cloudflare' => [
        'driver' => 'cloudflare',
        'account_id' => env('CLOUDFLARE_ACCOUNT_ID'),
        'queue_id' => env('CLOUDFLARE_QUEUE_ID'),
        'api_token' => env('CLOUDFLARE_API_TOKEN'),
    ],
],
```

11. Set `.env` variables:

```
QUEUE_CONNECTION=cloudflare
CLOUDFLARE_ACCOUNT_ID=your_account_id_here
CLOUDFLARE_QUEUE_ID=your_queue_id_here
CLOUDFLARE_API_TOKEN=your_token_here
```

---

3. Define How Jobs Are Processed

You must decide how queued messages will be executed:

- **If Laravel handles jobs (Pull Consumer):**
  - Decode JSON message → find job class → run `handle()` method.
- **If Cloudflare Worker handles jobs:**
  - Process messages in JavaScript.
  - Or call a Laravel API endpoint to execute heavy tasks.

Ensure you handle:

- Error handling and retries
- Dead-letter queues for failed jobs
- Idempotency (avoid duplicate effects)

---

4. Message Acknowledgment

Cloudflare Queues require explicit acknowledgment after a job succeeds.

If your consumer fails before acknowledgment, the message will be retried.
Cloudflare supports:

- Automatic retries
- Dead-letter queue for permanently failed messages
- Batch message processing

Docs: Cloudflare Queues API

---

5. Monitoring & Visibility

You should monitor:

- Number of pending and failed messages
- Processing latency
- Retry counts

Cloudflare Dashboard provides some metrics, but you can also log activity in Laravel or your Worker.

---

6. Gradual Migration Plan

To minimize risk, migrate in phases:

1. Start with small, non-critical jobs.
2. Keep Redis/Azure queues as fallback.
3. Observe performance and errors.
4. Gradually move all jobs once stable.

□ Example Implementation

Laravel Configuration Example

```
'cloudflare' => [
    'driver' => 'cloudflare',
    'account_id' => env('CLOUDFLARE_ACCOUNT_ID'),
    'queue_id'  => env('CLOUDFLARE_QUEUE_ID'),
    'api_token' => env('CLOUDFLARE_API_TOKEN'),
    'raw_handler' => CloudflareRawJobHandler::class,
],
```

This configuration uses the `ayles-software/laravel-cloudflare-queue` package to bridge Laravel's job dispatching with Cloudflare's Queues API.

Minimal Laravel Consumer (Pulling Messages)

1. Create a console command (`php artisan make:command CloudflareWork`).
2. In the `handle()` method:

```
public function handle()
{
    $queue = app('queue')->connection('cloudflare');
    while (true) {
        $job = $queue->pop();
        if ($job) {
            $job->fire();
              $queue->delete($job);
         }
         sleep(2);
     }
 }
```

This approach keeps the Laravel job logic intact, using Cloudflare as the message transport.

Cloudflare Worker Consumer (Alternative Approach)

If you want to use Workers directly:

```
export default {
  async queue(batch, env) {
    for (const message of batch.messages) {
      console.log('Received message:', message.body);
      // Optionally send to Laravel API for processing
      await fetch('https://your-laravel-app.com/api/run-job', {
        method: 'POST',
        body: JSON.stringify(message.body),
        headers: { 'Content-Type': 'application/json' },
      });
    }
  },
};
```

You can deploy this Worker in the **Cloudflare Dashboard → Workers → Create Worker**.

🚧 Common Challenges & Gotchas

Challenge	Description
Serialization mismatch	Laravel's job serialization doesn't directly work in Cloudflare Workers. Convert data to JSON.
Runtime limits	Workers have limited CPU time and memory. Avoid heavy jobs.
Retries & duplicates	Cloudflare may deliver a message more than once — ensure jobs are idempotent.
Error handling	Unhandled errors must be caught; use dead-letter queues for bad messages.
Monitoring	Need visibility into queue health, pending jobs, and failures.
Authentication	Secure your API tokens and Worker endpoints properly.

□ Recommended Action Plan

1. **Prototype** with one small Laravel job.
2. **Set up environment variables** and driver configuration.
3. **Push and verify** messages in Cloudflare Dashboard.
4. **Implement a basic consumer** (either Laravel script or Worker).
5. **Add retry & logging**.
6. **Gradually migrate** other job types.

🧠 Note for Current Setup

Currently, Cloudflare Queues **is not available under the Free Tier**.
It requires a **paid Cloudflare Workers subscription** to create and use queues.
Therefore, practical implementation is pending until a paid plan is activated.
However, the full process, configuration, and feasibility have been explored.

✅ Summary

Task	Status
Studied Cloudflare Queues documentation	✅ Done
Explored Laravel integration options	✅ Done

Task	Status
Verified available Laravel packages	✅ Done
Tested feasibility with free tier	⚠ Not possible (requires paid plan)
Ready for implementation	✅ Once subscription is enabled

Prepared by: *Yogesh Yadav*
Topic: Porting Laravel Queues to Cloudflare Queues
Date: *October 2025*
