DocsCrQueue

September 4, 2026

DocsCrQueue

Kind: Service

Source: `atloria-monorepo/apps/api/src/change-request/docs-cr-queue.service.ts`

The 'docs-cr' queue registration. Present in EVERY api pod for ENQUEUE (the request path
only adds a job) — api pods cannot touch the git repo (the RWO PVC is mounted only on the
worker), so every CR git operation travels through this queue to CrWorker, gated by DOCS_REPO_WORKER. Mirrors the docs-repo enqueue/worker split exactly.

Jobs are DEDUPED per change request via a deterministic jobId (`cr-open-<crId>`): a retried
POST for one CR collapses to a single pending job. Never blocks boot — an unavailable queue
just means the caller's enqueue returns false.

removeOnComplete AND removeOnFail are BOTH `true` (NOT {age,count}): a RETAINED completed OR
failed job with the same jobId makes BullMQ silently ignore a later add(), blocking a re-drive of the same CR (e.g. re-opening after a transient failure was fixed). Removing both
immediately keeps coalescing (a WAITING/ACTIVE job still absorbs duplicates) while the CR row
status ('draft'/'failed') owns retry semantics.

`DocsCrQueue` registers and manages the queue used for documentation change-request Git operations. API pods enqueue open, merge, refresh, or restage jobs without accessing the Git repository directly; the worker pod consumes those jobs when `DOCS_REPO_WORKER` is enabled.

Jobs are deduplicated per change request using deterministic job IDs, while completed and failed jobs are removed immediately so a CR can be re-driven after a transient failure.

Methods

Method	Signature	Returns	Description
`onModuleInit`	`onModuleInit()`	`void`	
`enqueueOpen`	`enqueueOpen(crId: string)`	`Promise<boolean>`	Enqueue the branch-staging step for a change request.
`enqueueMerge`	`enqueueMerge(crId: string)`	`Promise<boolean>`	Enqueue the merge-writeback step (row must already be atomically flipped to 'merging').
`enqueueRefresh`	`enqueueRefresh(crId: string)`	`Promise<boolean>`	Enqueue a diffCache/conflictState recompute against CURRENT main (status unchanged).
`enqueueRestage`	`enqueueRestage(crId: string)`	`Promise<boolean>`	Enqueue the restage of a pendingResolution onto the CR branch (resolver-authored commit).
`onModuleDestroy`	`onModuleDestroy()`	`Promise<void>`	

Dependencies

- `ConfigService`

Where it refuses work

- `DocsCrQueue` stops the work with an early return when `!this.queue`.

When something fails

- `DocsCrQueue` handles failure in 2 places: it logs it and continues in 1, and turns it into a return value in 1.

Diagram

```
sequenceDiagram
  participant Client
  participant API as API Pod
  participant Queue as DocsCrQueue
  participant Redis as BullMQ / Redis
  participant Worker as CrWorker
  participant Repo as Docs Git Repository

  Client->>API: POST change request action
  API->>Queue: enqueueOpen / enqueueMerge / enqueueRefresh / enqueueRestage
  Queue->>Redis: add(job, { jobId: deterministic CR ID })

  alt Matching waiting or active job exists
    Redis-->>Queue: Deduplicate job
    Queue-->>API: true
  else Queue available
    Redis-->>Queue: Job queued
    Queue-->>API: true
  else Queue unavailable
    Queue-->>API: false
  end

  Worker->>Redis: Consume queued job
  Worker->>Repo: Perform CR Git operation
  Worker->>Redis: Complete or fail job
  Redis->>Redis: Remove completed/failed job immediately
```

Usage

```
import { Injectable } from '@nestjs/common';
import { DocsCrQueue } from './docs-cr-queue.service';

@Injectable()
export class ChangeRequestService {
  constructor(private readonly docsCrQueue: DocsCrQueue) {}

  async openChangeRequest(crId: string): Promise<void> {
    // Persist the CR state first. The database row owns retry/status semantics.
    // The queue only schedules Git work for the worker pod.
    const queued = await this.docsCrQueue.enqueueOpen(crId);

    if (!queued) {
      // Do not fail the request solely because Redis/BullMQ is unavailable.
      // The CR remains in its draft/retryable state for later recovery.
      console.warn(`Unable to enqueue docs CR open job for ${crId}`);
    }
  }
```

```
  async mergeChangeRequest(crId: string): Promise<void> {
    const queued = await this.docsCrQueue.enqueueMerge(crId);

    if (!queued) {
      throw new Error(`Could not schedule merge processing for CR ${crId}`);
    }
  }
}
```

AI Coding Instructions

- Keep Git repository access exclusively in `CrWorker`; API pods should only persist CR state and enqueue jobs through `DocsCrQueue`.
- Use the existing deterministic job ID pattern for every CR action so duplicate requests collapse while a job is `WAITING` or `ACTIVE`.
- Preserve `removeOnComplete: true` and `removeOnFail: true`; retaining terminal jobs with the same `jobId` prevents BullMQ from accepting future re-drive attempts.
- Treat `false` enqueue results as queue availability failures, not as a reason to block NestJS startup or corrupt CR state.
- Ensure worker-side processing is gated by `DOCS_REPO_WORKER` and remains compatible with the action names emitted by `enqueueOpen`, `enqueueMerge`, `enqueueRefresh`, and `enqueueRestage`.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `ChangeRequestModule` (MODULE_PROVIDES)
- `ChangeRequestModule` (MODULE_EXPORTS)
- `ChangeRequestService` (DEPENDS_ON)