DocsRepoQueue

September 4, 2026

DocsRepoQueue

Kind: Service

Source: `atloria-monorepo/apps/api/src/docs-repo/docs-repo-queue.service.ts`

The 'docs-repo' queue registration. Present in EVERY api pod for ENQUEUE (the request path
only adds a job). Processing lives in CommitterWorker, gated by DOCS_REPO_WORKER so only a
dedicated 1-replica worker deployment commits — mirroring the technical-docs enqueue/worker
split.

Jobs are DEDUPED per project via a deterministic jobId (`commit-<projectId>`): a burst of
publish events for one project collapses to a single pending job, which the committer then
coalesces into one commit. 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(), dropping the next
project's commit (completed side) or blocking a once-poisoned project from ever being re-driven
after its cause is fixed (failed side). Removing both immediately keeps coalescing (a
WAITING/ACTIVE job still absorbs duplicates) while the outbox (attempts + poller) owns retry.

`DocsRepoQueue` registers the BullMQ queue used to enqueue documentation repository commit work from every API pod. It only adds deduplicated jobs; actual processing is

performed by the dedicated `CommitterWorker` deployment when `DOCS_REPO_WORKER` is enabled. Queue failures never block application startup or publishing flows—enqueue methods return `false` when the queue is unavailable.

Methods

Method	Signature	Returns	Description
`onModuleInit`	`onModuleInit()`	`void`	
`enqueueCommit`	`enqueueCommit(projectId: string)`	`Promise<boolean>`	Enqueue a coalesced commit for a project.
`enqueueBackfill`	`enqueueBackfill(projectId: string)`	`Promise<boolean>`	S5 prep: enqueue a one-shot backfill of a project's un-tagged PUBLISHED versions.
`onModuleDestroy`	`onModuleDestroy()`	`Promise<void>`	

Dependencies

- `ConfigService`

Where it refuses work

- `DocsRepoQueue` stops the work with an early return when `!this.queue`, in 2 places.

When something fails

- `DocsRepoQueue` handles failure in 3 places: it turns it into a return value in 2, and logs it and continues in 1.

Diagram

```
sequenceDiagram
  participant API as API Pod
  participant Queue as DocsRepoQueue
  participant Redis as BullMQ / Redis
  participant Worker as CommitterWorker
  participant Repo as Docs Repository

  API->>Queue: enqueueCommit(projectId)
```

```
  Queue->>Redis: add("commit", data, jobId: "commit-<projectId>")
  alt Queue available
    Redis-->>Queue: job accepted or deduplicated
    Queue-->>API: true
  else Queue unavailable
    Queue-->>API: false
  end

  Worker->>Redis: claim pending commit job
  Redis-->>Worker: coalesced project job
  Worker->>Repo: generate and commit documentation changes
  Worker->>Redis: complete job
  Note over Redis: removeOnComplete: true<br/>removeOnFail: true
```

Usage

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

@Injectable()
export class PublishService {
  constructor(private readonly docsRepoQueue: DocsRepoQueue) {}

  async publishProject(projectId: string): Promise<void> {
    // Persist publish state first, then request asynchronous docs synchronization.
    const queued = await this.docsRepoQueue.enqueueCommit(projectId);

    if (!queued) {
      // Do not fail publishing solely because Redis/BullMQ is unavailable.
      // The outbox or polling mechanism should retry later.
      console.warn(`Docs repository commit was not queued for ${projectId}`);
    }
  }

  async backfillProjectDocs(projectId: string): Promise<void> {
    const queued = await this.docsRepoQueue.enqueueBackfill(projectId);

    if (!queued) {
      console.warn(`Docs backfill was not queued for ${projectId}`);
    }
  }
}
```

AI Coding Instructions

- Use `enqueueCommit()` for normal publish-driven synchronization and `enqueueBackfill()` only for explicit regeneration/backfill workflows.

- Preserve deterministic per-project job IDs such as `commit-<projectId>` so bursts of events collapse into one waiting or active job.
- Do not move commit-processing logic into this service or API pods; processing belongs in `CommitterWorker`, gated by `DOCS_REPO_WORKER`.
- Keep `removeOnComplete` and `removeOnFail` set to `true`; retained terminal jobs with the same job ID prevent BullMQ from accepting future work.
- Treat a `false` enqueue result as non-fatal. Retry ownership belongs to the outbox, attempts, and polling flow rather than request handling.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `DocsRepoModule` (MODULE_PROVIDES)
- `DocsRepoModule` (MODULE_EXPORTS)
- `OutboxPollerService` (DEPENDS_ON)
- `SnippetsService` (DEPENDS_ON)