OpsAlertService

September 4, 2026

OpsAlertService

Kind: Service

Source: `atloria-monorepo/apps/api/src/technical-docs/ops-alert.service.ts`

Minimal ops alerting for pipeline failures. Always ERROR-logs; additionally POSTs to ALERT_WEBHOOK_URL when configured (Slack/Teams/generic — payload is `{ text }`).
Best-effort by design: alerting must never throw into the pipeline it reports on.

`OpsAlertService` provides minimal operational alerting for pipeline failures in the API backend. It always writes an `ERROR` log entry and, when `ALERT_WEBHOOK_URL` is configured, also sends a best-effort POST to that webhook using a simple `{ text }` payload compatible with Slack/Teams/generic receivers. Alerting is designed to never throw or disrupt the pipeline it is reporting on.

Methods

Method	Signature	Returns
`jobFailed`	`jobFailed(kind: string, projectId: string, error: string, jobId: string	null)`

Dependencies

- `ConfigService`

Where it refuses work

- `OpsAlertService` stops the work with an early return when `!this.webhookUrl`.

When something fails

- `OpsAlertService` handles failure in 1 place: it logs it and continues in all 1.

Diagram

```
sequenceDiagram
  autonumber
  participant Pipeline as Pipeline/Job Runner
  participant Service as OpsAlertService
  participant Logger as Nest Logger
  participant Webhook as ALERT_WEBHOOK_URL (Slack/Teams/etc.)

  Pipeline->>Service: alert(text)
  Service->>Logger: error(text)
  alt ALERT_WEBHOOK_URL configured
    Service->>Webhook: POST { text }
    Webhook-->>Service: 2xx/4xx/timeout
    Service->>Logger: (optional) debug/warn on delivery failure
  else not configured
    Service->>Logger: (optional) debug: webhook disabled
  end
  Service-->>Pipeline: resolve (never throws)
```

Usage

```
import { Injectable } from '@nestjs/common';
import { OpsAlertService } from './technical-docs/ops-alert.service';

@Injectable()
export class PipelineRunner {
  constructor(private readonly opsAlert: OpsAlertService) {}

  async runPipeline(pipelineId: string) {
    try {
      // ... pipeline work
      throw new Error('Step "build" failed with exit code 1');
    } catch (err: any) {
      const msg = `[pipeline:${pipelineId}] Failure: ${err?.message ?? String(err)}`;

      // Best-effort: logs ERROR; posts to ALERT_WEBHOOK_URL if configured; will not throw.
      await this.opsAlert.alert(msg);

      // Re-throw (or handle) independently of alerting.
      throw err;
    }
  }
}
```

AI Coding Instructions

- Keep alerting **best-effort**: never let network/webhook failures throw into callers; swallow/handle errors internally.
- Always emit an **ERROR-level log** for the alert message; webhook delivery is additive and optional.
- When posting to the webhook, send a minimal payload `{ text: string }` and avoid service-specific fields unless explicitly required.
- Treat `ALERT_WEBHOOK_URL` as an integration point: if unset/empty, skip HTTP calls and return quickly.
- Avoid blocking critical paths: use timeouts/retries conservatively (or none) so alerting cannot stall the pipeline.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `TechnicalDocsGenerationService` (DEPENDS_ON)
- `TechnicalDocsQueue` (DEPENDS_ON)
- `TechnicalDocsModule` (MODULE_PROVIDES)