Postmortem IR-2: rubygems.org rails application has stopped responding

Colby Swandale | Unrestricted access

> ✨ This draft postmortem contains AI-generated content that aims to help you as an author. This information is a starting point for your data gathering and cannot replace engineering analysis of the incident and its causes and solutions. All times are in UTC.

INCIDENT PROPERTIES	
Authors	Bits AI
Incident	IR-2
Severity Level	SEV-II (High) This incident was classified as SEV-II due to multi-region impact and reduced service capacity
Start Customer Impact	Jan 15, 7:21 am UTC
End Customer Impact	Jan 15, 7:28 am UTC

Executive Summary

Between Jan 15, 7:21 am UTC and Jan 15, 7:28 am UTC, users of the rubygems.org Rails application across multiple regions experienced reduced capacity and slow response times. This impact was caused by a 3-4x spike in traffic to the OpenSearch cluster, triggered by malformed search queries—primarily from suspected bot traffic—that were passed directly to OpenSearch without validation or sanitisation.

System Overview

- rubygems.org Rails application
  - Functionality: This service provides the main web application interface and API endpoints for rubygems.org, including handling user search queries.
  - Relationship with Other Services: It interacts directly with the OpenSearch cluster to process and return search results to users.
  - Assumptions: It was assumed that the application would handle search queries efficiently and that user-provided input would not cause excessive load or errors in downstream services.
- OpenSearch cluster
  - Functionality: This service indexes and searches gem data, executing queries received from the rubygems.org Rails application.
  - Relationship with Other Services: It serves as the backend search engine for the Rails application, processing all search requests and returning results.
  - Assumptions: It was assumed that the OpenSearch cluster would reliably process queries within expected timeframes and that the queries received would be well-formed and within operational limits.

Key Timeline

- Jan 15, 7:21 am UTC: Automated monitoring detects that the application is not responding within acceptable thresholds.
- Jan 15, 7:22 am UTC: Detection method is confirmed as originating from a monitoring system.
- Jan 15, 7:24 am UTC: Monitoring alert indicates the application is not reachable from multiple AWS regions.
- Jan 15, 7:31 am UTC: Incident is discussed; the application is intermittently responsive, with suspicion that the issue may recur until the search controller is patched.
- Jan 15, 7:35 am UTC: Initial summary identifies malformed search queries from suspected bot traffic causing a 3-4x spike in OpenSearch cluster traffic, resulting in degraded application performance and brief unavailability.
- Jan 15, 7:39 am UTC: Impact window is established: reduced application capacity and slow response from Jan 15, 7:21 am UTC to Jan 15, 7:28 am UTC.
- Jan 15, 7:42 am UTC: Root cause identified as unvalidated user-provided query strings passed directly to OpenSearch, allowing expensive and malformed queries.
- Jan 15, 7:50 am UTC: Incident state transitions to stable as immediate impact subsides.
- Jan 15, 7:59 am UTC: Work begins on a fix, focusing on query sanitisation and timeout adjustments.
- Jan 15, 8:00 am UTC–Jan 15, 8:10 am UTC: Investigation into timeout configurations for Searchkick and OpenSearch, with discussion of appropriate values and retry logic.
- Jan 15, 5:16 pm UTC: A branch is prepared to update the OpenSearch timeout to a more reasonable value, with local testing performed.
- Jan 15, 10:16 pm UTC–Jan 15, 10:23 pm UTC: Decision is made to set a 2-second timeout based on observed p95 search response times.
- Jan 15, 10:45 pm UTC: Plan shifts to setting the timeout at the OpenSearch client level for reliability.
- Jan 15, 11:46 pm UTC: Timeout configuration fix is deployed to production.
- Jan 16, 1:02 am UTC–Jan 16, 5:55 am UTC: Additional fix for search query sanitisation is developed and deployed to production, and the incident is marked as resolved.

Technical details

The incident was triggered by a surge in malformed search queries, likely from automated bot traffic, which exploited the lack of validation in the search API. An example query recorded from our Search Logs:

```
aws-sdk AND updated:[2025-06-18 TO *} AND updated:[2025-09-14 TO *} ...
```

Root Cause Analysis: Input validation was not implemented on search endpoints due to the assumption that only legitimate search patterns would be submitted. These queries, containing redundant and invalid date range filters, were passed directly to OpenSearch, causing excessive load and parse errors. The resulting spike in resource consumption led to degraded application performance and brief unavailability.

Performance Impact:

- P90 response time spiked from ~150ms to 20 seconds during the incident
- OpenSearch response time went from ~7ms to 70ms
- Search request rate (1-2 req/sec→ 10 req/sec)

Mitigation & Resolution: Mitigation involved implementing a 2-second timeout at the OpenSearch client level and deploying comprehensive query sanitisation to prevent similar abuse. These fixes were deployed within 24 hours of root cause identification, with performance returning to baseline (~150ms P90) by Jan 16, 5:55 am UTC.

Customer Impact

Starting at Jan 15, 7:21 am UTC and for approximately 6 minutes, all users of the rubygems.org Rails application experienced degraded performance, with the service operating at reduced capacity and slow to respond. During this period, the application was intermittently unavailable from multiple geographic locations, as detected by primary uptime monitors. The impact was limited to search functionality and related API endpoints, with no evidence of complete service outage.

Lessons Learned

What went well?

- We investigated and declared an incident
  - The page was automatically resolved by Datadog within a few seconds, which prevented this from being dismissed as a quirk—something that likely would have happened in the previous on-call setup
- We identified the problem & deployed fixes in less than 24 hours
- We implemented multiple fixes to address this from different levels to ensure we properly resolve this incident

What didn't go so well?

- Selecting the severity level was a bit confusing, Colby Swandale wasn't sure which one best matched the situation
  - In the moment, having clear and played out definitions of what each one means would help a lot
- We didn't get an alert about the malicious search queries before the incident occurred

Where did we get lucky?

- The incident only caused reduced operating capacity instead of a full outage of the Rails app

Did anything surprise us?

- The team was surprised that OpenSearch Query language could be passed directly through the Search Endpoints, effectively creating a query injection vulnerability similar to SQL injection or prompt injection attacks

Action Items

Planned:

- Update the on-call guides to document what types of incidents best align to what severity level in Datadog

- Restart work on adding the Fastly WAF to rubygems.org to capture these types of requests in the future and install automation rules escalate them to the team
- Investigate adding rate limiting for the Search endpoints (UI & API)
- Add alerting to OpenSearch when search queries are exceeding p99/p95 thresholds, CPU, memory & thread pool saturation
- Consider implementing search query cache & page cache

Completed Actions:

- Configured Searchkick timeout (2 seconds)
  - Add searchkick timeout of 2 seconds **MERGED**
- Implemented search query filtering and sanitization
  - Search query sanitisation **MERGED**