Elasticsearch at Fermilab

Kevin Retzke

30 Sept 2019

Overview

- Fermilab has one production Elasticsearch cluster, operated by the Scientific Computing Division “Landscape” project for grid, job, and data transfer monitoring.
- 19 data nodes (old grid workers) with ~ 100 TiB
- Three dedicated masters
- Two dedicated “client” nodes – all queries and ingest goes through these
- Two “frontend” nodes (behind HAProxy) with:
  - Grafana (six instances for different user groups)
  - Kibana
  - GraphQL API - “Lens”
  - HTTP data collection endpoint - “Ingest”
  - Apache httpd proxy handling routing and Shibboleth (SAML) SSO authentication
  - Graphite time-series database
  - Prometheus (internal service monitoring)

Deployment

- Elasticsearch 6.8 basic (free) license
- Deployed in Docker containers with docker-compose, local disk bind-mounted into container
- 40 TiB NAS disk NFS mounted for daily snapshots
- Index lifecycle maintenance with Curator
- Things we'd like to test:
  - Native lifecycle management
  - Rollup API

Security

- ReadOnlyRest (free license) on client nodes
  - Write access limited to Landscape-operated nodes
  - Read access on-site
  - Kibana admin and write access via LDAP groups
- httpd/shibboleth proxy limits Kibana access to logged-in users
- iptables limits in-cluster communication to Elasticsearch nodes
- Things we'd like to test/experiment with:
  - ES native security now that it's included in basic
  - Open Distro - https://opendistro.github.io/for-elasticsearch/

Data Sources

- HTCondor
  - Job, slot, other classads - condorbeat
  - event log – filebeat
  - job history – filebeat
- IFDH (data movement client) events - rsyslog
- SLURM jobcomp history - ingest
- Rucio transfer and deletion events – ingest
- dCache billing events – direct to Kafka
- Service container logs - logspout

Data Pipelines

- All data goes into “ingest” Kafka topics
  - Some services report directly
  - Beats go through logstash server (moving direct to Kafka)
  - External services talk to “Ingest” public HTTP service (next slide)
- Data processing with logstash or Python apps using Faust library.
- Enriched data goes to “digest” topics
- Logstash “store” processes read from Kafka and write to Elasticsearch (or other)
- All data pipeline services are run on single machine in Docker containers with docker-compose. Moving to OpenShift/OKD Kubernetes cluster soon
- Kafka cluster is three old grid workers, Kafka and Zookeeper running in Docker containers

Ingest Service

- Libbeat-based server that implements a limited set of Elasticsearch API write endpoints
- Allows services that can already talk to elasticsearch (e.g. SLURM) use that functionality, but gives us fine-grained control over where the data goes
- Simple HTTP API for others to implement

```
r.Path("/").Methods("GET").HandlerFunc(bt.pingHandler)
r.Path("/").Methods("HEAD").HandlerFunc(bt.headHandler)
r.Path("/_bulk").Methods("GET", "POST").HandlerFunc(bt.bulkHandler)
r.Path("/{index}").Methods("HEAD").HandlerFunc(bt.headHandler)
r.Path("/{index}/_bulk").Methods("GET", "POST").HandlerFunc(bt.bulkHandler)
r.Path("/{index}/{type}/_bulk").Methods("GET", "POST").HandlerFunc(bt.bulkHandler)
r.Path("/{index}/{type}").Methods("HEAD").HandlerFunc(bt.headHandler)
r.Path("/{index}/{type}").Methods("POST").HandlerFunc(bt.indexHandler)
r.Path("/{index}/{type}/").Methods("POST").HandlerFunc(bt.indexHandler)
r.Path("/{index}/{type}/{id}").Methods("PUT").HandlerFunc(bt.indexHandler)
r.Path("/{index}/_mapping/{type}").Methods("HEAD").HandlerFunc(bt.headHandler)
r.Path("/_xpack").Methods("GET").HandlerFunc(bt.xpackHandler)
```

- Logstash process routes data to Kafka topics based on index and type that data was written to

Data Access

- Primary use of data is curated Grafana dashboards
- Kibana for data exploration and ad-hoc visualization
  - Users can request access to save visualizations
- Direct read access to Elasticsearch allowed but discouraged outside Landscape
- GraphQL API provides programmatic access to job data without needing to know Elasticsearch topic and field details – allows us to change or move data

Lens GraphQL API

- Written in Go with gqlgen https://gqlgen.com/
- In-memory shared cache of Elasticsearch queries using groupcache
- Web-based schema documentation and query builder (public) https://landscape.fnal.gov/lens
- Combines data from several index patterns. Minimum queries are made to provide only the data requested
- Allows us to change index patterns, mapping, fields, etc. without affecting users
- Success story: allowed POMS (production job workflow management tool) to remove all job monitoring and job status database

Lens API Example

Query

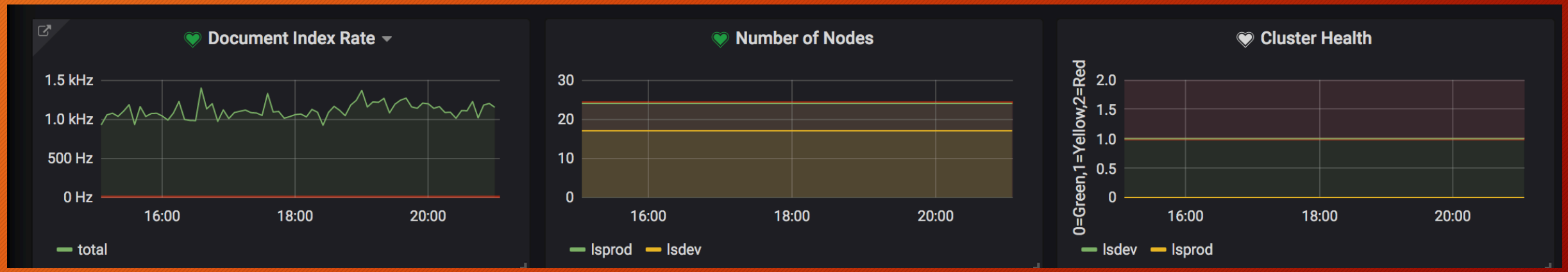
```
{
  submissions(group:"nova") {
    id
    done
    jobs(limit:10){
      id
      status
    }
  }
}
```

Response


```
{
  "data": {
    "submissions": [
      {
        "id": "24010071@jobsub01.fnal.gov",
        "done": false,
        "jobs": [
          {
            "id": "24010071.0@jobsub01.fnal.gov",
            "status": "running"
          },
          {
            "id": "24010071.100@jobsub01.fnal.gov",
            "status": "running"
          },
```

Elasticsearch Monitoring and Alerting

- Cluster health and status collected with Prometheus exporter https://github.com/Braedon/prometheus-es-exporter
- Collected by Prometheus servers running on “frontend” nodes
- Monitoring in Grafana dashboards with alerts on key metrics



Issues

- Mysterious unresponsive master nodes causing monitoring (including Kibana!) to timeout (on 5, not seen since upgrade to 6)
- Painful and time-consuming upgrade from 5 to 6 due to many breaking changes (mainly mapping type removal). Expect 7 will be just as bad since some deprecated behavior is being removed.
- Nodes crashing (SIGILL) after restart while loading index state off disk (ongoing)
- Mapping explosions (notably with job classads)
- I miss joins 😿