title: Elasticsearch Monitoring Integration description: Collect and monitor key Elasticsearch metrics such as request latency, indexing rate, and segment merges with built-in anomaly detection, threshold, and heartbeat alerts. Send notifications to email and various chatops messaging services, correlate events & logs, filter metrics by server, node, time or index, and visualize your cluster's health with out of the box graphs and custom dashboards

Agent Install

Setting up the monitoring agent takes less than 5 minutes:

1. Create an Elasticsearch App in the Integrations / Overview (or Sematext Cloud Europe). This will let you install the agent and control access to your monitoring and logs data. The short What is an App in Sematext Cloud video has more details.
2. Name your Elasticsearch monitoring App and, if you want to collect Elasticsearch logs as well, create a Logs App along the way.
3. Install the Sematext Agent according to the setup instructions displayed in the UI.
4. Enable HTTP metrics by setting `http.enabled: true` and set the `node.name` value in `elasticsearch.yaml`.

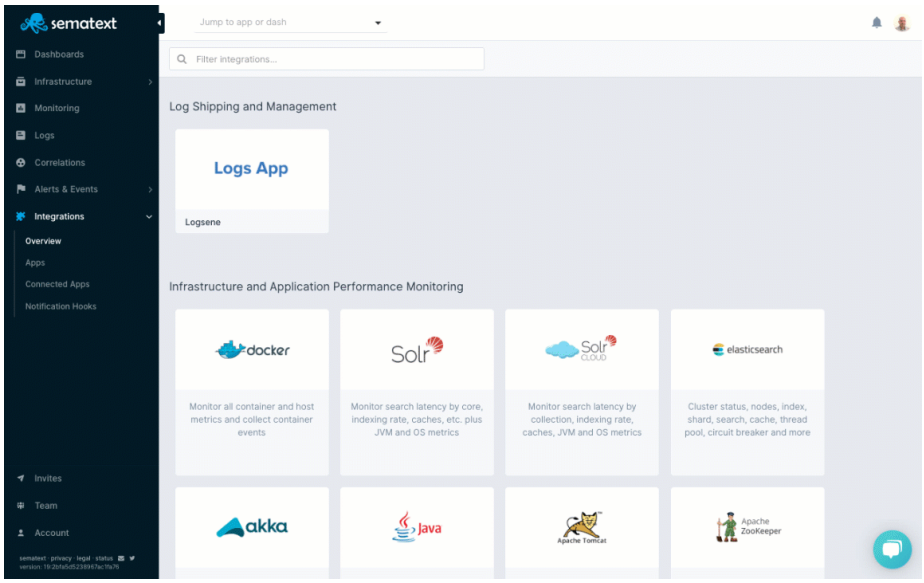


Figure 1: App creation and setup instructions in Sematext Cloud

For example, on Ubuntu, add Sematext Linux packages with the following command:

```
echo "deb http://pub-repo.sematext.com/ubuntu sematext main" | sudo tee /etc/apt/sources.lis
```

```
wget -O - https://pub-repo.sematext.com/ubuntu/sematext.gpg.key | sudo apt-key add -
sudo apt-get update
sudo apt-get install sematext-agent
```

Then setup Elasticsearch monitoring by providing Elasticsearch server connection details:

```
sudo bash /opt/spm/bin/setup-sematext  \
    --monitoring-token <your-token-goes-here>   \
    --app-type elasticsearch  \
    --agent-type standalone  \
    --ST_MONITOR_ES_NODE_HOSTPORT 'localhost:9200'
```

Make sure that HTTP metrics are enabled by setting `http.enabled: true` in `elasticsearch.yaml`. Also set the `node.name` value in the same file. Elasticsearch will otherwise generate a random node name each time an instance starts, making tracking node stats over time impossible.

The `elasticsearch.yml` file can be found in `/etc/elasticsearch/elasticsearch.yml` or `$ES_HOME/config/elasticsearch.yml`.

Important Metrics to Watch and Alert on

System and JVM Metrics

The first place we would recommend looking for in a new system are the OS metrics: CPU, memory, IO and network. A healthy CPU graph looks like this:

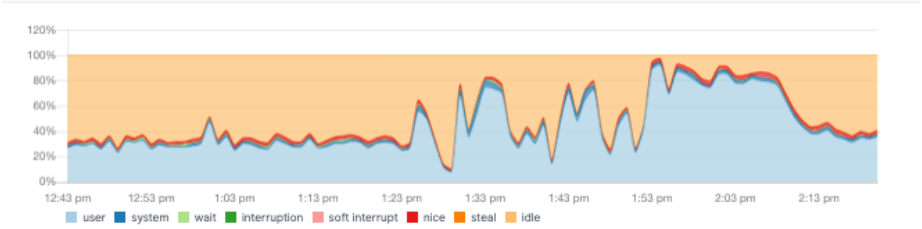


Figure 2: CPU usage

Note how the relative percentage of *wait* and *system* is negligible compared to *user*. Meaning we don't have a bottleneck in IO. And total usage isn't close to 100% all the time, so there's headroom.

If there's high CPU usage, have a look at JVM garbage collection (GC) times. Which are probably good candidates for alerts. If GC times are high, then Elasticsearch is in trouble with JVM memory, rather than doing useful work with the CPU. You can look deeper into JVM memory usage to check. A healthy pattern looks like a shard tooth:

Figure 3: JVM memory usage per pool

When it comes to system memory, don't be worried if you see very little free, like here:

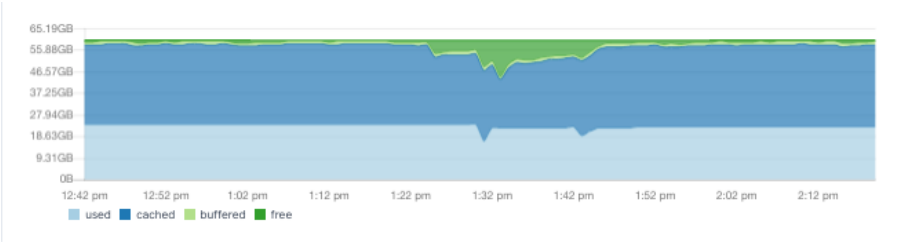


Figure 4: System memory usage

The operating system will try to cache your index files as much as it can. The *cached* memory can be freed up, if the system needs more memory. ### Elasticsearch-specific metrics You'll want to monitor query rates and times. In other words, how fast is Elasticsearch responding? Since this will likely impact your users, these are metrics worth alerting on as well.

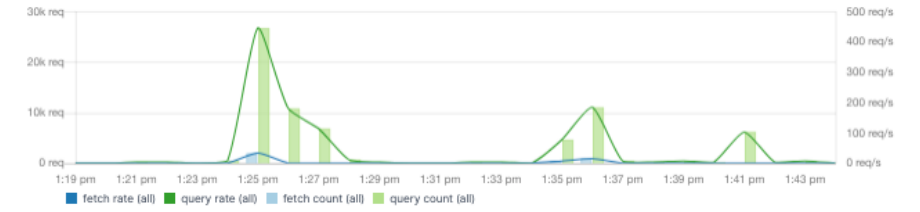
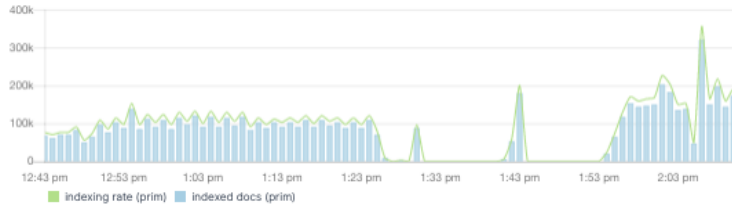


Figure 5: Query and fetch rate



On the indexing side, check the indexing rate:

And correlate it with the asynchronous refresh and merge times, as they can correlate with your CPU spikes:

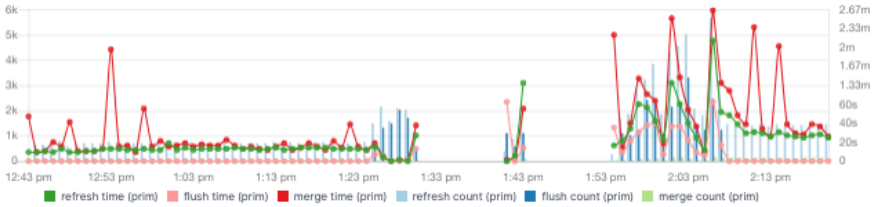


Figure 6: Refresh, flush and merge stats

For example, if refresh time is too high, you might want to adjust the refresh interval.

Last, but certainly not least, you may want to get an alert if a node leaves the cluster, so you can replace it. Once you do, you can keep an eye on shard stats, to see how many are initializing or relocating:

Alert Setup

There are 3 types of alerts in Sematext:

- **Heartbeat alerts**, which notify you when a Elasticsearch DB server is down
- Classic **threshold-based alerts** that notify you when a metric value crosses a predefined threshold
- Alerts based on statistical **anomaly detection** that notify you when metric values suddenly change and deviate from the baseline

Let's see how to actually create some alert rules for Elasticsearch metrics in the animation below. The request query count chart shows a spike. We normally have up to 100 requests, but we see it can jump to over 600 requests. To create an alert rule on a metric we'd go to the pulldown in the top right corner of a chart and choose "Create alert". The alert rule applies the filters from the current view and you can choose various notification options such as email or configured notification hooks (PagerDuty, Slack, VictorOps, BigPanda, OpsGenie, Pusher, generic webhooks etc.)

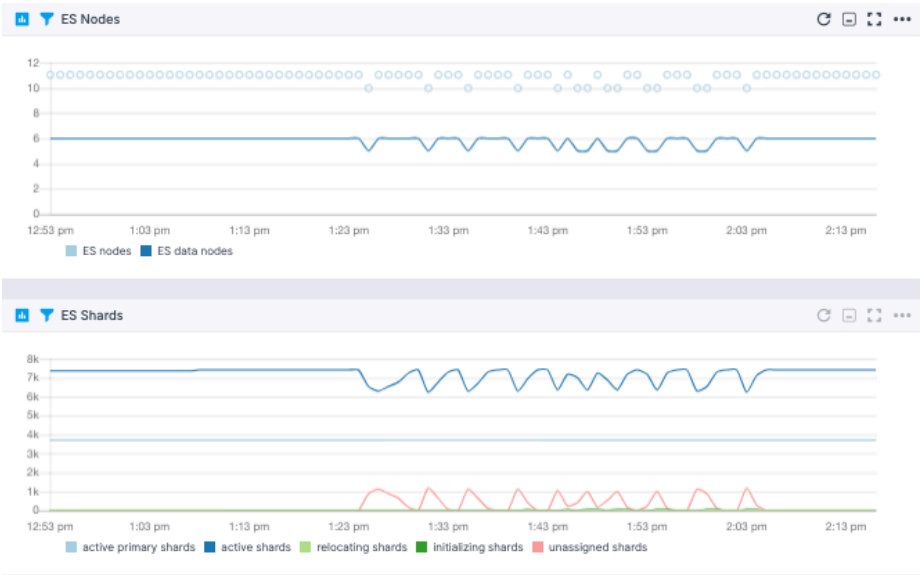


Figure 7: Dropping nodes and relocation of shards

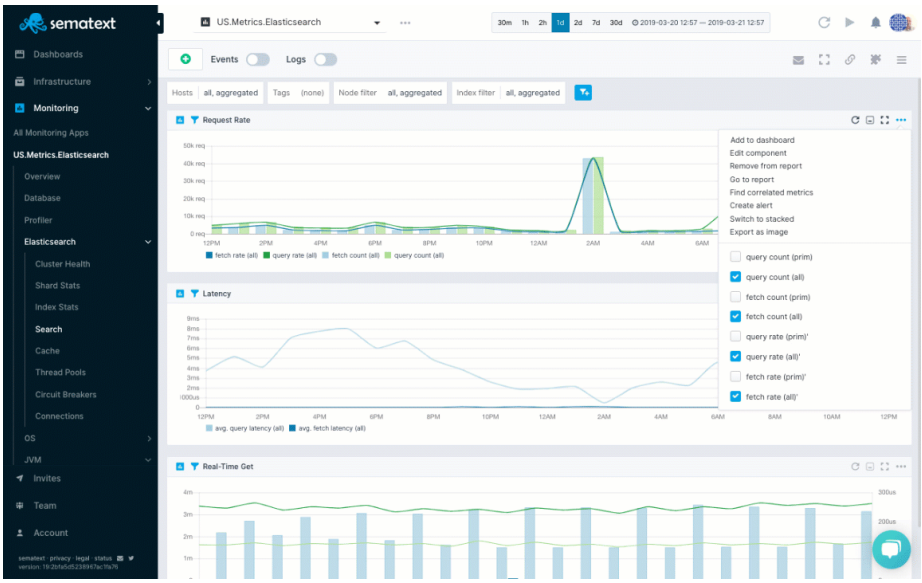


Figure 8: Alert creation for Elasticsearch request query count metric

Correlating Logs and Metrics

Since having logs and metrics in one platform makes troubleshooting simpler and faster let's ship Elasticsearch logs too. You can use many log shippers, but we'll use Logagent because it's lightweight, easy to set up, and because it can parse and structure logs out of the box. ### **Shipping Elasticsearch Logs** 1. Create a Logs App to obtain an App token 2. Install Logagent npm package

```
sudo npm i -g @sematext/logagent
```

you don't have Node.js, you can install it easily. E.g. On Debian/Ubuntu:

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
sudo apt-get install -y nodejs
```

3. Install the Logagent service by specifying the logs token and the path to Elasticsearch log files. You can use `-g 'var/log/**/elasticsearch*.log`` to ship only logs from Elasticsearch server. If you run other services, on the same server consider shipping all logs using `-g `/var/log/**/*.log`` The default settings ship all logs from /var/log/**/*.log when the -g parameter is not specified. Logagent detects the init system and installs Systemd or Upstart service scripts. On Mac OS X it creates a launchd service. Simply run:

```
sudo logagent-setup -i YOUR_LOGS_TOKEN -g `var/log/**/elasticsearch*.log`
#for EU region:
#sudo logagent-setup -i LOGS_TOKEN
#-u logsene-receiver.eu.sematext.com
#-g `var/log/**/elasticsearch*.log`
```

The setup script generates the configuration file in /etc/sematext/logagent.conf and starts Logagent as system service.

Log Search and Dashboards

Once you have logs in Sematext you can search through them when troubleshooting, save queries you run frequently or create your individual logs dashboard.

Elasticsearch Metrics and Log Correlation

A typical troubleshooting workflow starts from detecting a spike in the metrics, then digging into logs to find the root cause of the problem. Sematext makes this really simple and fast. Your metrics and logs live under the same roof. Logs are centralized, the search is fast, and the powerful log search syntax is simple to use. Correlation of metrics and logs is literally one click away.

More about Elasticsearch Monitoring

- Elasticsearch Monitoring Guide
- Top 10 Elasticsearch Metrics To Monitor

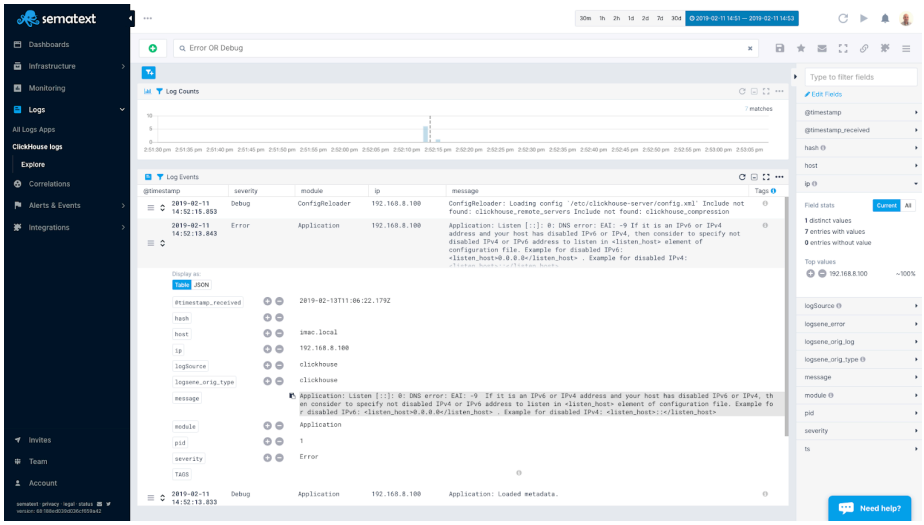


Figure 9: Search for Elasticsearch Logs

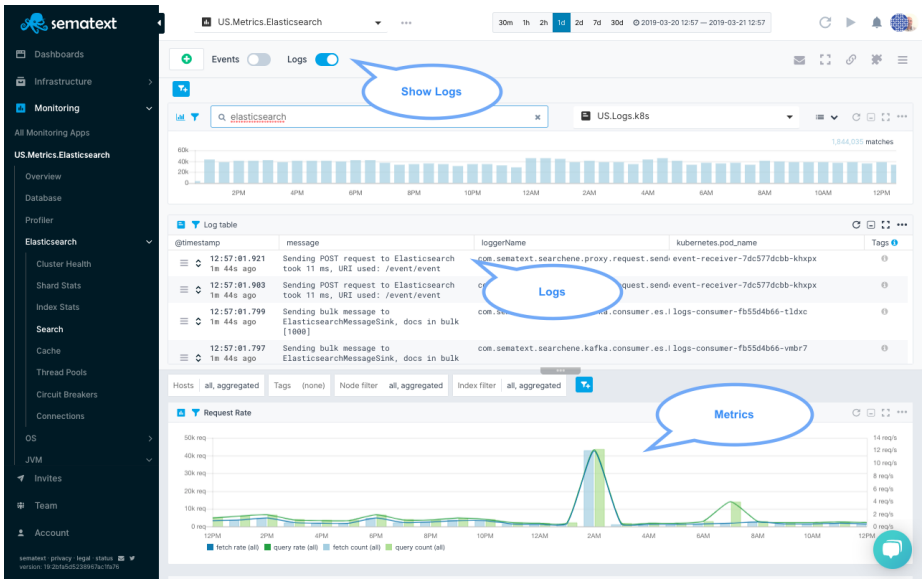


Figure 10: Elasticsearch logs and metrics in a single view

- Elasticsearch Open Source Monitoring Tools
- Monitoring Elasticsearch With Sematext

Metrics

Metric Name Key *(Type) (Unit)*	Description
fieldData max size**es.circuitBreaker.fieldData.size.max** *(long gauge) (bytes)*	max fieldData size
fieldData estimated size**es.circuitBreaker.fieldData.size.estimate** *(long gauge) (bytes)*	estimated fieldData size
fieldData overhead**es.circuitBreaker.fieldData.size.overhead** *(double gauge)*	fieldData overhead
request maximum size**es.circuitBreaker.request.size.max** *(long gauge) (bytes)*	max request size
request estimated size**es.circuitBreaker.request.size.estimate** *(long gauge) (bytes)*	estimated request size
request overhead**es.circuitBreaker.request.size.overhead** *(double gauge)*	request overhead
ES nodes**es.cluster.nodes** *(long gauge)*	Number of nodes in the ES cluster
ES data nodes**es.cluster.nodes.data** *(long gauge)*	Number of data nodes in the ES cluster
active primary shards**es.cluster.health.shards.active.primary** *(long gauge)*	Number of active primary shards
active shards**es.cluster.health.shards.active** *(long gauge)*	Number of active shards
relocating shards**es.cluster.health.shards.relocating** *(long gauge)*	Number of currently relocating shards
initializing shards**es.cluster.health.shards.initializing** *(long gauge)*	Number of currently initializing shards
unassigned shards**es.cluster.health.shards.unassigned** *(long gauge)*	Number of currently unassigned shards

Metric Name Key *(Type) (Unit)*	Description
open HTTP connns**es.connection.http.current.open** *(long gauge)*	open HTTP conns (current_open)
total opened HTTP connns**es.connection.http.total.opened** *(long gauge)*	total opened HTTP conns (total_opened)
open TCP connns**es.connection.tcp.server.open** *(long gauge)*	open TCP conns (server_open)
network received packets**es.transport.rx.packets** *(long counter)*	network received packets count (rx_count)
network received size**es.transport.rx.bytes** *(long counter) (bytes)*	network received size (rx_size)
network transmitted packets**es.transport.tx.packets** *(long counter)*	network transmitted packets count (tx_count)
network transmitted size**es.transport.tx.bytes** *(long counter) (bytes)*	network transmitted size (tx_size)
active conn openings**es.connection.tcp.active.opens** *(long counter)*	active conn openings (active_opens)
passive conn openings**es.connection.tcp.passive.opens** *(long counter)*	passive conn openings (passive_opens)
open sockets**es.connection.tcp.current.estab** *(long gauge)*	open sockets (current_estab)
inbound segments (in_segs)**es.connection.in.segs** *(long counter)*	inbound segments (in_segs)
outbound segments (out_segs)**es.connection.out.segs** *(long counter)*	outbound segments (out_segs)
retransmitted segments (retrans_segs)**es.connection.retrans.segs** *(long counter)*	retransmitted segments (retrans_segs)
socket resets (estab_resets)**es.connection.tcp.estab.resets** *(long counter)*	socket resets (estab_resets)
failed socket open (attempt_fails)**es.connection.tcp.attempt.fails** *(long counter)*	failed socket open (attempt_fails)

Metric Name Key *(Type) (Unit)*	Description
connection errors**es.connection.in.errors** *(long counter)*	connection errors
socket resets sent (out_rsts)**es.connection.tcp.out.rsts** *(long counter)*	socket resets sent (out_rsts)
docs count (prim)**es.index.docs.primaries** *(long gauge)*	docs count on primary shards
docs deleted (prim)**es.index.docs.deleted.primaries** *(long gauge)*	docs deleted on primary shards
docs count (all)**es.index.docs.totals** *(long gauge)*	docs count on all (primary and replica) shards
docs deleted (all)**es.index.docs.deleted.total** *(long gauge)*	docs deleted on all (primary and replica) shards
size on disk (prim)**es.index.files.size.primaries** *(long gauge) (bytes)*	size on the disk of primary shards
size on disk (all)**es.index.files.size.total** *(long gauge) (bytes)*	size on the disk of all (primary and replica) shards
indexed docs (prim)**es.indexing.docs.added.primaries** *(long counter)*	docs indexed on primary shards
deleted docs (prim)**es.indexing.docs.deleted.primaries** *(long counter)*	docs deleted on primary shards
indexing time (prim)**es.indexing.time.added.primaries** *(long counter) (ms)*	time spent indexing on primary shards
deleting time (prim)**es.indexing.time.deleted.primaries** *(long counter) (ms)*	time spent deleting on primary shards
indexed docs (all)**es.indexing.docs.added.total** *(long counter)*	docs indexed on all (primary and replica) shards
deleted docs (all)**es.indexing.docs.deleted.total** *(long counter)*	docs deleted on all (primary and replica) shards
indexing time (all)**es.indexing.time.added.total** *(long counter) (ms)*	time spent indexing on all (primary and replica) shards

Metric Name Key *(Type) (Unit)*	Description
deleting time (all)**es.indexing.time.deleted.total** *(long counter) (ms)*	time spent deleting on all (primary and replica) shards
gc collection count**jvm.gc.collection.count** *(long counter)*	count of GC collections
gc collection time**jvm.gc.collection.time** *(long counter) (ms)*	duration of GC collections
open files**jvm.files.open** *(long gauge)*	jvm currently open files
max open files**jvm.files.max** *(long gauge)*	jvm max open files limit
used**jvm.pool.used** *(long gauge) (bytes)*	jvm pool used memory
used**jvm.pool.max** *(long gauge) (bytes)*	jvm pool max memory
thread count**jvm.threads** *(long gauge)*	current jvm thread count
peak thread count**jvm.threads.peak** *(long gauge)*	peak jvm thread count
merge count (prim)**es.indexing.merges.primaries** *(long counter)*	merge count on primary shards
merge time (prim)**es.indexing.merges.time.primaries** *(long counter) (ms)*	merge time on primary shards
merged docs count (prim)**es.indexing.merges.docs.primaries** *(long counter)*	merged docs count on primary shards
merged docs size (prim)**es.indexing.merges.docs.size.primaries** *(long counter) (bytes)*	merged docs size on primary shards
merge count (all)**es.indexing.merges.total** *(long counter)*	merge count on all (primary and replica) shards
merge time (all)**es.indexing.merges.time.total** *(long counter) (ms)*	merge time on all (primary and replica) shards
merged docs count (all)**es.indexing.merges.docs.total** *(long counter)*	merged docs count on all (primary and replica) shards
merged docs size (all)**es.indexing.merges.docs.size.total** *(long counter) (bytes)*	merged docs size on all (primary and replica) shards
field cache evictions**es.cache.field.evicted** *(long counter)*	Field cache evictions
field cache size**es.cache.field.size** *(long gauge) (bytes)*	Field cache size
filter cache evictions**es.cache.filter.evicted** *(long counter)*	Filter cache evictions

Metric Name Key *(Type) (Unit)*	Description
filter cache size**es.cache.filter.size** *(long gauge) (bytes)*	Filter cache size
warmer current**es.cache.warmer.current** *(long gauge)*	Warmer current
warmer total**es.cache.warmer.total** *(long counter) (bytes)*	Warmer total
warmer total time**es.cache.warmer.time** *(long counter) (ms)*	Warmer total time
filter/query cache count**es.cache.filter.size.count** *(long counter)*	Filter/query cache count of elements
refresh count (prim)**es.indexing.refreshes.primaries** *(long counter)*	refresh count on primary shards
refresh time (prim)**es.indexing.refreshes.time.primaries** *(long counter) (ms)*	refresh time on primary shards
refresh count (all)**es.indexing.refreshes.total** *(long counter)*	refresh count on all (primary and replica) shards
refresh time (all)**es.indexing.refreshes.time.total** *(long counter) (ms)*	refresh time on all (primary and replica) shards
flush count (prim)**es.indexing.flushes.primaries** *(long counter)*	flush count on primary shards
flush time (prim)**es.indexing.flushes.time.primaries** *(long counter) (ms)*	flush time on primary shards
flush count (all)**es.indexing.flushes.total** *(long counter)*	flush count on all (primary and replica) shards
flush time (all)**es.indexing.flushes.time.total** *(long counter) (ms)*	flush time on all (primary and replica) shards
query count (prim)**es.query.count.primaries** *(long counter)*	query count on primary shards
query latency (prim)**es.query.latency.time.primaries** *(long counter) (ms)*	query latency on primary shards
fetch count (prim)**es.fetch.count.primaries** *(long counter)*	fetch count on primary shards

Metric Name Key *(Type) (Unit)*	Description
fetch latency (prim)**es.fetch.latency.time.primaries** *(long counter) (ms)*	fetch latency on primary shards
avg. query latency (primaries)**es.query.latency.primaries.avg** *(long gauge) (ms)*	avg. query latency on primary shards
query count (all)**es.query.count.total** *(long counter)*	query count on all (primary and replica) shards
query latency (all)**es.query.latency.time.total** *(long counter) (ms)*	query latency on all (primary and replica) shards
fetch count (all)**es.fetch.count.total** *(long counter)*	fetch count on all (primary and replica) shards
fetch latency (all)**es.fetch.latency.time.total** *(long counter) (ms)*	fetch latency on all (primary and replica) shards
avg. query latency (all)**es.query.latency.total.avg** *(long gauge) (ms)*	avg. query latency on all (primary and replica) shards
real-time get count (prim)**es.request.rtg.primaries** *(long counter)*	real-time get count on primary shards
real-time get latency (prim)**es.request.rtg.time.primaries** *(long counter) (ms)*	real-time latency on primary shards
real-time get exists count (prim)**es.request.rtg.exists.primaries** *(long counter)*	real-time get exists count on primary shards
real-time get exists latency (prim)**es.request.rtg.exists.time.primaries** *(long counter) (ms)*	real-time get exists latency on primary shards
real-time get missing count (prim)**es.request.rtg.missing.primaries** *(long counter)*	real-time get missing count on primary shards
real-time get missing latency (prim)**es.request.rtg.missing.time.primaries** *(long counter) (ms)*	real-time get missing latency on primary shards
real-time get count (all)**es.request.rtg.total** *(long counter)*	real-time get count on all (primary and replica) shards
real-time get latency (all)**es.request.rtg.time.total** *(long counter) (ms)*	real-time latency on all (primary and replica) shards

Metric Name Key *(Type) (Unit)*	Description
real-time get exists count (all)**es.request.rtg.exists.total** *(long counter)*	real-time get exists count on all (primary and replica) shards
real-time get exists latency (all)**es.request.rtg.exists.time.total** *(long counter) (ms)*	real-time get exists latency on all (primary and replica) shards
real-time get missing count (all)**es.request.rtg.missing.total** *(long counter)*	real-time get missing count on all (primary and replica) shards
real-time get missing latency (all)**es.request.rtg.missing.time.total** *(long counter) (ms)*	real-time get missing latency on all (primary and replica) shards
active shards**es.cluster.shards.active** *(long gauge)*	Number of active shards
active primary shards**es.cluster.shards.active.primary** *(long gauge)*	Number of active primary shards
initializing shards**es.cluster.shards.initializing** *(long gauge)*	Number of initializing shards
relocating shards**es.cluster.shards.relocating** *(long gauge)*	Number of relocating shards
unassigned shards**es.cluster.shards.unassigned** *(long gauge)*	Number of unassigned shards
active threads**es.thread.pool.active** *(long gauge)*	active threads
thread pool size**es.thread.pool.size** *(long gauge)*	thread pool size
thread pool queue**es.thread.pool.queue** *(long gauge)*	thread pool queue
thread pool queue size**es.thread.pool.queue.size** *(long gauge)*	thread pool queue size
rejected threads**es.thread.pool.rejected** *(long counter)*	rejected threads
thread pool largest**es.thread.pool.largest** *(long gauge)*	thread pool largest
completed threads**es.thread.pool.completed** *(long counter)*	complete threads
thread pool min**es.thread.pool.min** *(long gauge)*	thread pool min

Metric Name Key *(Type) (Unit)*	Description
thread pool max**es.thread.pool.max** *(long gauge)*	thread pool max

FAQ

** Why doesn't the number of documents I see in SPM match the number of documents in my Elasticsearch index **

SPM collects index stats from primary shards only. To see the total number of documents in an index, select all shards in that index and choose "sum". The list of shards and the "sum" function can be found in the "Shard filter" in the Index Stats report.

** Can Sematext Agent collect metrics even when Elasticsearch HTTP API is disabled **

Each Sematext Agent collects Elasticsearch metrics only from the local node by accessing the Stats API via HTTP. To allow only local access add the following to elasticsearch.yml. Don't forget to restart each ES node to whose elasticsearch.yml you add this.

```
http.host: "127.0.0.1"
```

** Can I point Sematext Agent to a non-localhost Elasticsearch node **

Yes. Adjust /opt/spm/spm-monitor/conf/spm-monitor-config-*TOKEN_HERE*-default.properties and change the SPM_MONITOR_ES_NODE_HOSTPORT property from the default localhost:9200 value to use an alternative hostname:port. After that restart Sematext Agent (if you are running a standalone App Agent version) or Elasticsearch process(es) with embedded App Agent.

** My Elasticsearch is protected by basic HTTP authentication, can I use Sematext Agent **

Yes. You just need to adjust /opt/spm/spm-monitor/conf/spm-monitor-config-*TOKEN_HERE*-default.properties file by adding the following two properties (replace values with your real username and password):

```
ST_MONITOR_ES_NODE_BASICAUTH_USERNAME=yourUsername
ST_MONITOR_ES_NODE_BASICAUTH_PASSWORD=yourPassword
```

Restart your Sematext Agent after this change (either with **sudo service spm-monitor restart** in case of standalone App Agent or by restarting Elasticsearch node if you are using in-process App Agent).

** I am using Sematext Agent and I don't see Index (and/or Refresh/Flush/Merge) stats, why is that **

Sematext Agent collects Index stats only from primary shards, so it is possible that you installed Sematext Agent on some Elasticsearch node which hosts only

replicas. The same is also true for Refresh/Flush and Merge stats. Also note that Sematext Agent should be installed on all your Elasticsearch nodes to get the complete picture of your cluster in SPM Reports UI.