Elasticsearch internals

Martin Toshev
@martin_fmi

Agenda

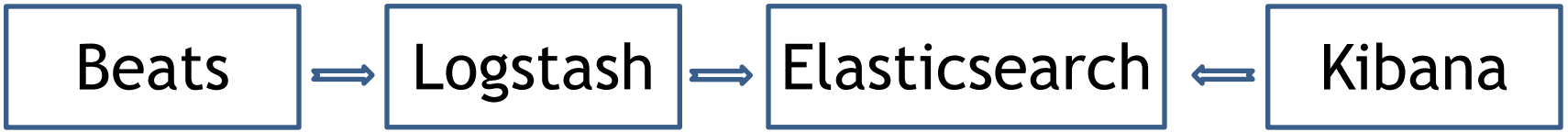
- The ELK stack
- Elasticsearch architecture
- Extending Elasticsearch

The ELK stack

Overview

- The ELK stack is centered around Elasticsearch and includes:
  - Elasticsearch
  - Kibana
  - Logstash
  - Beats
- Elasticsearch is a full-text search engine with numerous capabilities that drive it as a leading solution on the market

Overview



data collection

data aggregation and processing

Indexing and storage

analysis and visualization

Elasticsearch

- A web server build on top of the Lucene Java library
- Another way to describe it is a document-oriented database
- Provides more functionality not provided by Lucene such as:
  - Caching
  - Clustering
  - JSON-based REST API

Elasticsearch

- The basic data structure used by Elasticsearch is an **inverted index**
- Indexes are stored on disk in separate files
- Search can be performed on multiple indexes
- Documents in an index are logically grouped by type (deprecated in 7.0.0)

Elasticsearch

- In order to ensure result relevancy Elasticsearch uses a few algorithms to calculate relevant scores

- The default one used is **tf-idf** (**term frequency-inverse document frequency**)

Elasticsearch

- Provides faster retrieval of documents for more scenarios than a traditional RDBMS
- A traditional RDBMS uses indexes implemented using a B-tree or hash table structures
- The RDBMS indexes pose significant limitations on the types of queries where they can be applied

Elasticsearch

- Documents might not have an explicit schema specified
- An explicit schema (mapping) for certain fields can be specified
- Certain fields can match a pattern that identifies their field types (dynamic mapping)
- The same field can be indexed multiple times using different mechanisms

Kibana

- An analytics and visualization dashboard designed to work with Elasticsearch
- Provides a number of additional tools used to simplify interaction with Elasticsearch
- Browser-based interface

Logstash

- Data collection engine with pipeline processing capabilities
- Provides integration with a variety of third-party data sources
- Originally targeted for log collection but for used for a variety of use cases in practice

Logstash

- Data is processed in an input-filter-output manner
- Output data is stored typically in an Elasticsearch index
- Plug-in architecture with a large number of third-party plug-ins available

Beats

- Beats is a collective name for a set of log shippers
- Each beat is able to collect logs from a particular third-party source
- They are lightweight in nature and are installed separately
- Data collected from the various beats can be sent to Elasticsearch or Logstash

Catalog of beats

Beat	Use
Auditbeat	Collects audit data
Filebeat	Collects log files
Functionbeat	Collects cloud data
Heartbeat	Collects data from availability monitoring
Journalbeat	Collects data from systemd journals
Metricbeat	Collects system-level metrics
Packetbeat	Collects network traffic
Winlogbeat	Collects windows event logs

X-Pack

- X-Pack is a set of extra features for the ELK stack
- Already installed by default with Elasticsearch
- To see a list of X-Pack features installed the _xpack endpoint can be used:

```
GET /_xpack
```

Elasticsearch architecture

Clustering

- Elasticsearch is designed with clustering in mind
- By default each node starts with 5 shards
- An Elasticsearch shard is a Lucene index

Clustering

- The more nodes are added to the cluster shards get distributed among nodes
- By default, Elasticsearch tries to balance the number of shards across your nodes so the load is evenly spread
- Partial results can be returned from shards that are still available

Clustering

- The shard for a document is determined based on the following formula:

```
shard = hash(<rounting_key>) % number_of_primary_shards
```

- By default the rounting key is the document ID

- A custom routing key can be set during indexing to enable shard routing

Clustering

- By default, new nodes discover existing clusters via multicast
- If a cluster is discovered, the new node joins it only if it has the same cluster name
- If a node on the same instance already runs on the specified port Elasticsearch tries to pick the next available port

Clustering

- Two ways to discover nodes:
  - multicast: automatic discovery of nodes on the network, multicast ping send by default to 224.2.2.4:54328
  - unicast: explicitly specify cluster nodes in the Elasticsearch configuration

```
discovery.seed_hosts = ["10.0.0.3", "10.0.0.4:9300",
                        "10.0.0.5[9300-9400]"]
```

- In unicast discovery not all nodes need to be listed
- The ones that are listed need to know of the other nodes in the cluster

Scaling considerations

- When planning an Elasticsearch cluster several aspects need to be considered:
  - sharding
  - splitting data between indexes
  - maximizing throughout

Sharding considerations

- Too small number of shards introduces a scalability bottleneck
- Too many shards introduces performance and management overhead
- Determining the number of shards should be based on an upfront planning

Splitting data between indexes considerations

- Avoid putting huge amounts of data in a single index
- Index allocation strategies might be adopted such as a daily/ weekly/yearly index. For example: **orders-20200106**
- Aliases can be used to avoid changing the name of the index

Types of cluster nodes

- An Elasticsearch node can be of the following types:
  - **master** `node.master = true`
  - **data** `node.data = true`
  - **ingest** `node.ingest = true`

Concurrency control

- Elasticsearch uses optimistic locking for concurrency control
- When indexing a document the **version** attribute can be specified
- If the document already has the specified version the operation is rejected from Elasticsearch
- Concurrency control can also be achieved using **the if_seq_no** and **if_primary_term** parameters of an index request

High availability

- To increase availability, you can create one or more copies (called replicas) for each of your initial shards (called primaries)

- Once an indexing request is send to a particular shard (determined from a hash of the document's ID)
  the document being indexes is also sent to the shard's replicas

High availability

- When you perform a search request Elasticsearch distributes the load among the shards and the replicas

- In that manner replicas are also used to improve performance and not only provide a mechanism for fault-tolerance

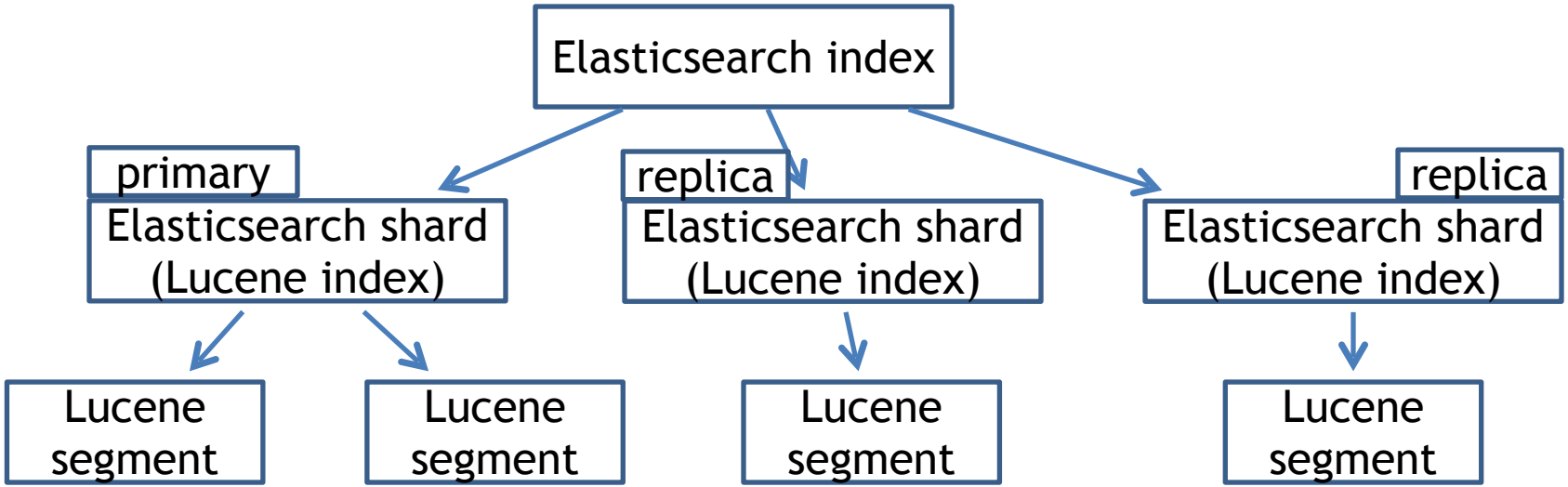
High availability

- In addition to shard replication Elasticsearch provides additional facilities for high availability in case replication is not sufficient:
  - cluster backup and restore
  - cross cluster replication

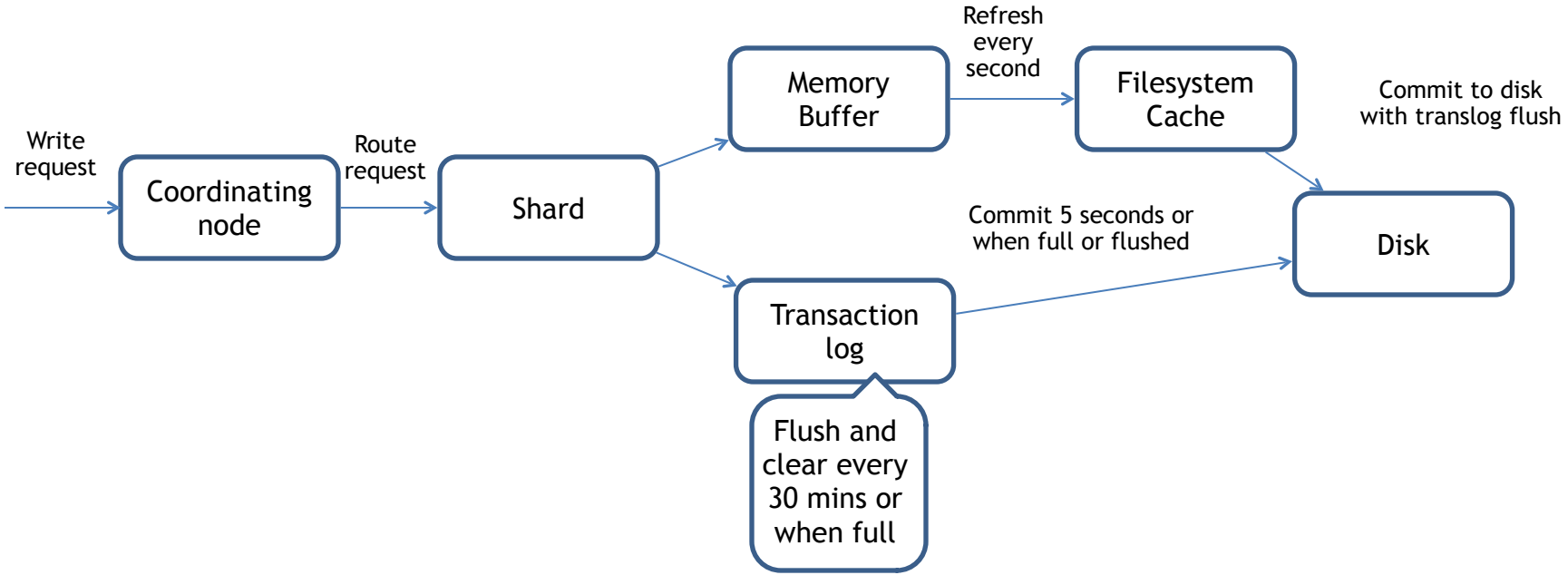
Elasticsearch in production

- General guidelines:
  - allocate enough heap memory for search operations (machines with 32-64GB are preferable)
  - prefer CPUs with more cores than faster CPUs, Elasticsearch utilizes the various cores by means of the distinct thread pools it uses
  - prefer faster storage systems such as SSDs if possible, no need to levarage RAID-based mirroring and parity in favor of shard replication
  - Use fast network for an Elasticsearch cluster

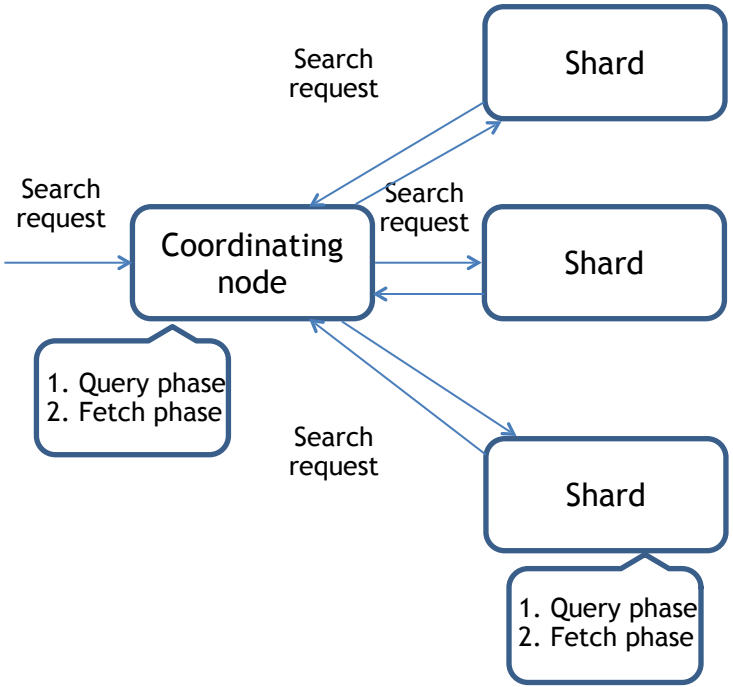
The Elasticsearch index



Shard request processing

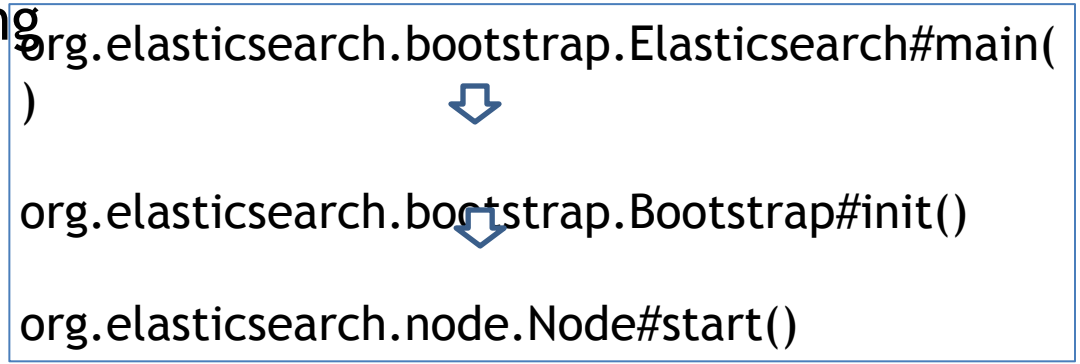


Shard request processing



Elasticsearch modules

- Internally Elasticsearch is comprised of different modules
- Modules are loaded during Elasticsearch instance startup
- Elasticsearch uses a modified version of Google Guice for the module binding



Elasticsearch modules

```
// b is a Guice binder
modules.add(b -> {
                b.bind(Node.class).toInstance(this);
                b.bind(NodeService.class).toInstance(nodeService);

b.bind(NamedXContentRegistry.class).toInstance(xContentRegistry);
                b.bind(PluginsService.class).toInstance(pluginsService);
                b.bind(Client.class).toInstance(client);
                b.bind(NodeClient.class).toInstance(client);
                b.bind(Environment.class).toInstance(this.environment);
                b.bind(ThreadPool.class).toInstance(threadPool);
                b.bind(NodeEnvironment.class).toInstance(nodeEnvironment);
            …
}
```

Elasticsearch modules

- Some core modules are:
  - **Discovery and cluster formation**: used for node discovery
  - **HTTP**: for the HTTP REST API
  - **Plugins**: for managing the Elasticsearch plug-ins
  - **Thread pools**: thread pools used internally by Elasticsearch
  - **Transport**: communication layer for the Elasticsearch nodes

The Elasticsearch codebase (demo)

Extending Elasticsearch

Elasticsearch plug-ins

- Plug-ins extend the functionality of Elasticsearch
- Located under the **plugins** directory
- The **elasticsearch-plugin** utility can be used to manage plugins

```
bin/elasticsearch-plugin install [core_plugin_name]
```

```
bin/elasticsearch-plugin install [URL]
```

```
bin/elasticsearch-plugin list
```

```
bin/elasticsearch-plugin remove [plugin_name]
```

Elasticsearch plug-ins

- Elasticsearch plug-ins are bundled in a ZIP archive along with their dependencies
- They are loaded by a separate classloader
- Distinct security permissions can be applied per plug-in

Elasticsearch plug-ins

- Elasticsearch plug-ins must implement the **org.elasticsearch.plugins.Plugin** class

- An instance of **org.elasticsearch.plugins.PluginService** is responsible to load plug-ins

Writing an Elasticsearch plug-in (demo)

Q&A

Thank you