Elasticsearch

Please see github for a more updated version of this doc https://github.com/sakaiproject/sakai/tree/master/search

Overview

The search implementation in Sakai has been rewritten backed with Elasticsearch. Elasticsearch is a multi-tenant, cloud friendly, search service written on top of Lucene. It has a JSON based REST api, and has been proven to scale extremely well with large datasets in clustered environments. Elasticsearch (ES) can run as a standalone server or embedded. The current implementation in Sakai embeds ES into Sakai. It would not be an huge effort to allow ES to run standalone and integrate with Sakai this way as well, but that work has not be completed to date.

The implementation is a drop in replacement for the legacy search. It aims to provide the same functionality as the previous search, but there are a few differences which are outlined in the Functional section that follows.

The code is in trunk here: https://source.sakaiproject.org/svn/search/trunk/elasticsearch/

One of the nice things about the JSON api over HTTP is that even when ES is embedded you can administer it from the REST interface. Most ES configuration is runtime configuration. There are several Javascript only frontends for ES. In addition simply curl commands can be used to change configuration. Configuration and the frontend tools are explained more the sections that follow.

If you would like an overview of how Search works from a user perspective, see our Help documentation.

Functional Information

Indexing

The new search does not replace the legacy components that provides what entities can be indexed in Sakai. It reuses the existing EntityContentProducer (ECP) mechanism. Indexing happens in one of three ways. Either a Site can be indexed, all sites can be indexed, or documents are indexed in real-time. The site and full index mechanisms are triggered from within the search admin tool. When a site is indexed each ECP is asked for a list of resources its aware of and then every document is scheduled for indexing in ES. This goes fairly quickly as it does not do full digesting of the content, but rather puts a placeholder in ES within the content. There is a bulk indexer thread that runs and queries for documents that do not have content and performs the more heavy weight digesting of content (turning bytes into text strings). Using this mechanism we allow every node in the cluster to share in the indexing load, which prevents any one node from being consumed by bulk indexing work, and speeds up full indexing time. There are configuration options that control how often this thread runs and how big the batch size is and how big each individual bulk request to ES is. These options are explained in more detail in the configuration section. The bulk index thread runs at a lower priority than user requests so even when bulk indexing is occurring it will not put an unreasonable amount of load on the system. User requests will be serviced first.

The full index follows the same pattern only every site that is indexable will have its resources scheduled for indexing.

The types of content that can be indexed has not changed and is summarized as follows:

TODO...

Searching

A normal search will match requests against the title and content of documents that have been indexed. In addition, users can pick to search in the current site or all sites. For every search result, the content that matches the query will be highlighted in bold as is common practice for most searches. Any content a user does not have access to will be search but will be filtered out of the results with a message that simply counts how many unaccessible documents where found. This is all consistent with how the old search functioned. In the future we may move to a model that better integrates ES with Sakai's ACLs, but for now this is how security is enforced.

There is also now an autocomplete search that matches the search query against titles that have been indexed. This operates in much the same way as google drive does. So as you type a drop down will appear that shows titles of documents that match. This is new functionality.

The old search also provided a "tag" tab that would show the term vectors for a search and show the more frequent terms in larger fonts. This feature can be enable in the new implementation but it is not recommended at this time. ES uses a mechanism called facetting to provide aggregation data like this. In practice is has been discovered that facetting adds a large memory requirement to the system as every doc that is found has to be put in memory to count up the matching terms. With a large dataset this will certainly cause problems. Lucene 4.0 and a new version of ES that uses it, is going to provide an opportunity for facetting to be refactored. Once this version is available we should be able to refactor the Sakai code to use it, and provide things like autocomplete, did you mean, and other common search features is a way that does put ridiculous memory requirements on the system. So it is highly recommended that useFacetting@org.sakaiproject.search.api.SearchService=false for production systems until a new version of ES is out that better supports these features in a way Sakai can consume.

You can also search specific content types, this is new functionality. This is similar to how gmail works. So if you only want to search chats, prepend chat: to your search "chat:elasticsearch" or for searching sites, "site:history 101".

General Requirements

Since ES runs embedded there aren't any new server components necessary. ES will need RAM, but effort has been done to limit its RAM consumption to the point that the big heaps typical Sakai installations are running with should be sufficient. You will need to either enable multicast in your network or if you prefer you unicast, you'll need to open up the ports between the nodes for communication. More detail on the options can be found in the sections that follow. Typically 9300 is used for node to node communication and 9200 is used for HTTP access to ES Rest api. You don't necessarily have to open 9200 and enable that, but it is recommended as it provides a lot of administration capability when you do. You'll want to make sure these ports are blocked from anything outside the internal network.

Sakai Configuration

Supported existing configuration

To turn search on:

```
search.enable = true
```

To only index sites that have the Search tool placed: (this is the default)

```
 onlyIndexSearchToolSites@org.sakaiproject.search.api.SearchIndexBuilder=true
```

To only exclude user sites from indexing (this is the default)

```
excludeUserSites@org.sakaiproject.search.api.SearchService=true
```

To configure an excluded site list which never get indexed (the default follows).

```
ignoredSites@org.sakaiproject.search.api.SearchService=~admin,!admin,PortfolioAdmin
```

New configuration

To enable facetting, which will allow the "tag" tab to work: (not recommended for production)

```
useFacetting@org.sakaiproject.search.api.SearchService=false
```

To turn off site filtering (this is on by default). You can decide to use filters to match against sites, or the query itself can list the sites. In theory, filters should be faster as the results of filters are cached in ES. This may require more memory to do so. In practice, the response times don't seem to differ much with this on or off.

```
useSiteFilters@org.sakaiproject.search.api.SearchService=false
```

To turn the autocomplete feature off, this is on by default:

```
useSuggestions@org.sakaiproject.search.api.SearchService=false
```

To control the size of the batch index thread. The larger this is the most likely it is that nodes will end up indexing the same thing wasting cycles. If it set to low indexing will be slow. In practice you want your bulk size to not take longer than how often you are running the bulk thread.

```
contentIndexBatchSize@org.sakaiproject.search.api.SearchIndexBuilder=100
```

The bulkRequestSize controls the number of requests that are rolled into one ES call. Settings this too low will cause a lot more merges and can slow things down or even cause data issues. Setting is too high is a memory concern as the docs will be in memory until they are flushed out. 10-20 is probably a reasonable number depending on the size of your docs.

```
bulkRequestSize@org.sakaiproject.search.api.SearchIndexBuilder=20
```

How often the bulk index job runs in seconds (default is every minute):

```
period@org.sakaiproject.search.api.SearchIndexBuilder=60
```

Elasticsearch Configuration

First of all a word about how ES configuration works in Sakai. ES has a bunch of configuration that is possible. Any ES configuration can be set by prepended "elasticsearch." to the property name and including in your sakai.properties file. There is also the ability to configure more detailed options concerned the document mapping and index configuration which typically take JSON strings.

The Basics

The path to store local index data (by default this is sakai.home/elasticsearch). Each node will append its own path to this, so if you are using a shared filesystem for index data you don't need to worry about making this unique on each node. Note each node has its own copy of the shards its working with.

```
elasticsearch.path.data=/files/elasticsearch
```

Shards and replicas. ES breaks the index into segments called shards. Typically, no single node would contain the full index but pieces of it. Replicas are how many copies of each shard to keep. The default for ES is 5 shards and 1 replica. Generally, you want to aim for around the same number of shards as you have nodes. You can't change the number of shards without doing a full re-index, so its best to plan for some growth. The number of replicas will increase your disk consumption and slow index time down a bit (allow this can be managed at runtime, see tuning section).

```
elasticsearch.index.number_of_shards=5
elasticsearch.index.number_of_replicas=1
```

Turning on http communication so you can use curl and other tools. You want to make sure this is firewalled to the outside world, but its really handy to have on even in production.

```
elasticsearch.http.enabled=true
elasticsearch.http.port=9200
```

Discovery

For detailed information on various options go read the discovery guide.

Multicast (the default)

Discovery is the way ES nodes find each other. Multicast is the default, if multicast is allowed in your environment there is actual nothing to do, it will just work. You don't have to worry about multiple clusters on the same network as long as your serverName are distinct, which they would really have to be anyway, unless you are using IPs. Sakai tells ES to use the serverName property to uniquely identify nodes in the cluster (in ES this is called the cluster.name), in this way you won't get nodes talking to the wrong cluster, and you really shouldn't have to modify anything for that to work, since you probably have the serverName's unique already.

Now if multicast is a problem, it is in AWS and a lot of IT orgs frown on it, then you need another solution. Next up unicast.

Unicast

With unicast you are basically going to call out the location of every node in your cluster so they allow know exactly how to talk to. This obviously requires a full cluster restart and isn't a terribly elastic solution, but for Sakai this is pretty typical and probably will work for a lot of people. At a minimum you'll need the following.

```
#turn multicast off
elasticsearch.discovery.zen.ping.multicast.enabled=false
# pick your communication port
elasticsearch.transport.tcp.port=9300
# set the location of all your nodes, including the port
elasticsearch.discovery.zen.ping.unicast.hosts=ec2-184-169-221-255.us-west-1.compute.amazonaws.com:9300,ec2-204-236-163-255.us-west-1.compute.amazonaws.com:9300,ec2-184-169-227-255.us-west-1.compute.amazonaws.com:9300,ec2-204-236-170-255.us-west-1.compute.amazonaws.com:9300
```

EC2

If you are in EC2 use can use unicast, but if you want things to be more elastic, you can use the EC2 discovery mechanism which will use a EC2 service for registration and eliminates the need to articulate the address of every node. The AWS dependency that is needed is already included in Sakai. For more detail, go here.

Inter-node communication

elasticsearch nodes communicate with each other using **both TCP and UDP** on port 9300 (or 9300, 9301, and upwards if you have more than one app server on the same server). It is important that these ports are unfirewalled between Sakai application servers.

If your application servers have more than one IP address, you may need to specify the IP address which elasticsearch should bind to and use to communicate with other nodes. You can do this by setting

```
elasticsearch.network.host=A.B.C.D
```

(where A.B.C.D is the sever's IP address) in an app-server-specific Sakai properties file such as security.properties. See Network Settings for more detail.

Mapping

The default mapping can be found in elasticsearch/impl/src/resources/org/sakaiproject/search/elastic/bundle/mapping.json. If you wish to change it, you can set in in sakai.properties like this:

```
mapping@org.sakaiproject.search.api.SearchIndexBuilder=                \
 {                                                                     \
    "sakai_doc": {                                                     \
        "_source": {                                                   \
            "enabled": false                                           \
        },
\                                                                      \
        "properties": {                                                \
            "siteid": {                                                \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "title": {                                                 \
                "type": "string",                                      \
                "store": "yes",                                        \
                "term_vector" : "with_positions_offsets",              \
                "search_analyzer": "str_search_analyzer",              \
                "index_analyzer": "str_index_analyzer"                 \
            },                                                         \
            "url": {                                                   \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "reference": {                                             \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "id": {                                                    \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "tool": {                                                  \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "container": {                                             \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "type": {                                                  \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "subtype": {                                               \
                "type": "string",                                      \
                "index": "not_analyzed",                               \
                "store": "yes"                                         \
            },                                                         \
            "contents": {                                              \
                "type": "string",                                      \
                "analyzer": "snowball",                                \
                "index": "analyzed",                                   \
                "store": "no"                                          \
            }                                                          \
        }                                                              \
    }                                                                  \
}
```

For detail on mapping.

Index Settings

The default index settings are configured by the elasticsearch/impl/src/resources/org/sakaiproject/search/elastic/bundle/indexSettings.json file. Simply index properties can be adjusted by the sakai.properties. For example, shards and replicas:

```
elasticsearch.index.number_of_shards=5
elasticsearch.index.number_of_replicas=1
```

There are number of things that can be controlled by the index settings. For some common ones go here. For detail on the settings concerning the various modules you can tweak go here.

For more complicated properties use the JSON method like this. Note, JSON properties are overridden by sakai.properties.

```
indexSettings@org.sakaiproject.search.api.SearchIndexBuilder=  \
{                                                              \
    "analysis": {                                              \
        "filter": {                                            \
            "substring": {                                     \
                "type": "nGram",                               \
                "min_gram": 2,                                 \
                "max_gram": 20                                 \
            }                                                  \
        },                                                     \
        "analyzer": {                                          \
            "snowball": {                                      \
                "type": "snowball",                            \
                "language": "English"                          \
            },                                                 \
            "str_search_analyzer": {                           \
                "tokenizer": "keyword",                        \
                "filter": ["lowercase"]                        \
            },                                                 \
            "str_index_analyzer": {                            \
                "tokenizer": "keyword",                        \
                "filter": ["lowercase", "substring"]           \
            }                                                  \
        }                                                      \
    }                                                          \
}
```

The nGram filter and last two analyzers are for the autocomplete feature search. The snowball analyzer is the normal analyzer for basic content searching.

Tuning and Recommendations

The aim of this whole project was to basically eliminate the index corruption that would send Sakai into useless indexing with the legacy search. The ideal is that you will do a bulk system reindex once and then never really have to do that again. The replicas are key to part of this. You want enough replicas so that if a node goes down or something happens to its disk, you can recover. There is obviously a storage consideration as well, the more replicas the more storage you'll need. Elasticsearch wants to store the whole source document by default, we've turned that off in our mapping as we already have the source in CHS, so there is no need to store it twice. Even still expect your index data to get large. Depending on the types of content in your system, you can probably expect the index without any replicas to be about 25-30% of your CHS storage (assuming everything in your CHS is indexable, which is probably not the case). With 1 replica you'd expect that to be about 50-60%.

In addition, there is the capability of a gateway which is like a backup plan for recovery if your index gets wacked or for when bringing large clusters online. There are several options for the gateway, shared storage, s3 extra. For more on that see the ES gateway guide.

It our testing on medium AWS instance with 8 GBs ram we are seeing the index process about 100 docs per minute. So a 4 node cluster can process around 100,000 docs in 4 hours. You can't exactly correlated the number of docs that get indexed to the raw number in CHS since not everything gets indexed. We are still working out some rules of thumb there, but the assumption is probably some on the order of 25% of repository is actually indexable. We will update this page with recommendations as better estimates are discovered.

When you go to do a full index a few things will help you out. First of all the number of replicas greatly slows down the indexing, as all those bytes need to be copied around. You can actually change this at runtime. So you can go for example with 0 replicas, wait for the indexing to finish and then go to 1 or 2 replicas. You can do that via curl like this:

No replicas

```
curl -XPUT 'http://localhost:9200/sakai_index/_settings' -d '{"index" : {"number_of_replicas" : 0}}'
```

1 replicas after indexing finishes

```
curl -XPUT 'http://localhost:9200/sakai_index/_settings' -d '{"index" : {"number_of_replicas" : 1}}'
```

We saw a 30-40% improvement in indexing time doing this.

In addition, you can change the index refresh rate at runtime, this gave us a 10% improvement. Like this:

No refresh during indexing:

```
curl -XPUT 'http://localhost:9200/sakai_index/_settings' -d '{"index" : {"index.refresh_interval" : -1}}'
```

Back to a reasonable refresh rate after indexing:

```
curl -XPUT 'http://localhost:9200/sakai_index/_settings' -d '{"index" : {"index.refresh_interval" : "5s"}}'
```

You might also want to play with the (contentIndexBatchSize@org.sakaiproject.search.api.SearchIndexBuilder) size and your thread timing (period@org.sakaiproject.search.api.SearchIndexBuilder). Smaller batch sizes running more often may improve performance.

The Elasticsearch dev list seems to recommend that that Java heap not take up all the memory on the box. I've read recommendations that you actually leave 50% of the ram for the os. This is so that file system caching and other low level things can use the ram. There is a lot of IO going on. I have not tried this recommendation yet but plan on it. We've been running tested on 8GB servers with 6GB for the tomcat heap. I'll update my findings if I can bring some more concrete data to the table.

The indexing does a lot of deletes and adds. Deletes in Lucene are not actually removed but simply flagged, this is for speed of the action. But this can lead to larger indices over time. After a bulk load you can optimize the index and remove deletes by issuing one or both of these commands. This action is an expensive IO operation and could time some time, but it greatly reduces the size of the index and speeds up searches as well.

```
curl -XPOST 'http://localhost:9200/sakai_index/_optimize?only_expunge_deletes=true'
curl -XPOST 'http://localhost:9200/sakai_index/_optimize?max_num_segments=1'
```

Tools

There are a number of integrations and front ends for elasticsearch these can be helpful for monitoring and administration. Things like elasticsearch-head are super helpful as its just a lightweight javascript app that you point to your HTTP endpoint. There is nothing to install in Sakai or anywhere to use it. For a decent list of option go here.

For unix admins fiddlers types, curl is your friend. You can do anything with ES with just curl.