Searching with Alfresco 5.1 Enterprise

Table of Contents

Author: Martin Bergljung

Introduction

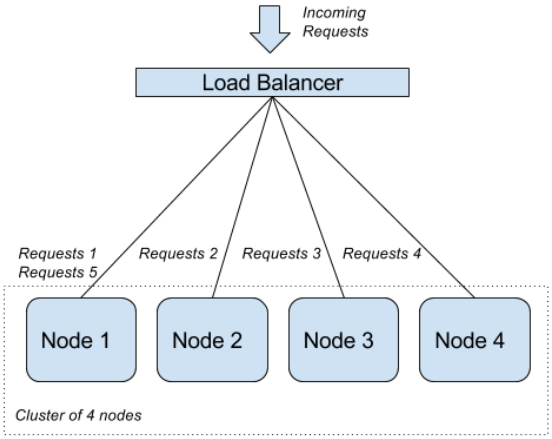
This article builds on the "Searching with Alfresco 5.1 Community" article, which covers all the standard basic stuff around searching in Alfresco with Solr, and extends it with information about how to scale your search solution when you are starting to reach tens of millions of documents.
When you are reaching in the area of 50 million content items, which should be indexed and searchable, you will start to see benefits from clustering your solution and splitting up the index between different nodes in the cluster.
In this article I will assume that you have been using Alfresco for a while, and that you have a production environment with millions of content items in the repository. And you are now starting to think about how to handle the growth of the content repository in the future. Maybe you even have some non-functional requirements saying that you should be able to support hundreds of millions of content items in the near future, with similar fast search experience as you have now.
I will not assume that you know anything about clustering, replication, sharding, distributed search, high availability etc. This is what this article is about. I do however assume that you are up to date with what was covered in the "Searching with Alfresco 5.1 Community" article.
When you install Alfresco it comes with Solr, which is used when you search the repository. It is however not configured in any clustering setup, just the basic Solr instance with two cores for the live content and the archived content. If you have a Community version of Alfresco you can't actually use any of the clustering features of Alfresco, you need an Enterprise license for that, and install the Enterprise version. That's why this article is called *Searching with Alfresco 5.1 Enterprise*.
Make sure to also check-out Alfresco Scalability Blueprint.

Scalability Concepts and Terminology

When we start talking about scalability, fault tolerance, high availability etc, then new concepts and terms that we don't normally hear will come up from time to time. It's good to get up to speed on these before diving into the details.
For the basic search concepts and terms see the first article. This section covers only stuff around scalability and the Enterprise version of Alfresco.

Cluster

A cluster is most likely something you have already heard about and you might even have experience setting up an Alfresco cluster. A cluster is basically a way of running multiple application servers on different machines in parallel. The reason you do this is to support more users (i.e. higher load) and to provide a more fault tolerant solution.
To talk to the servers you usually go through a load balancer, which directs you to the appropriate server based on some algorithm such as round robin (i.e. just take the next available in order), weighted (take the one with the least load), etc. The load balancer can also usually detect if a server is down and then skip using it. The load balancer talks to the servers via a LAN.
Each machine in a cluster is usually referred to as a node (not to be confused with an Alfresco Repository node…). You can say you got a 4-node cluster for example, meaning you got 4 identical servers (as far as functionality goes) running in parallel with a load balancer in front of them:



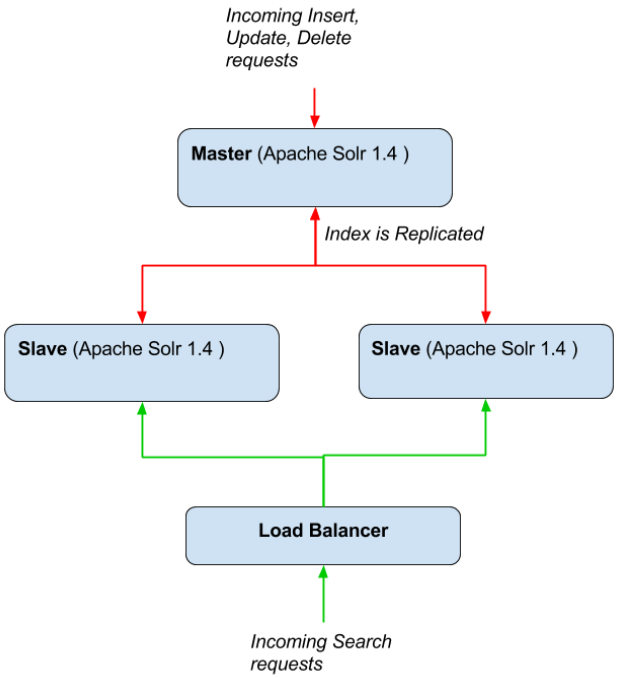
To the end-user a cluster is just viewed as a single system providing a specific service, such as search.

Replication

Replication is used to improve reliability, fault-tolerance, and accessibility. The replication itself is usually transparent to the end-user. And in a failure scenario, a failover of replicas is hidden as much as possible. We talk about active and passive replication in systems that replicate data or services:

- **active replication** is performed by processing the same request at every replica.
- **passive replication** involves processing each single request on a single replica and then transferring its resultant state to the other replicas.

Solr 1.4 uses passive replication with the Master-Slave architecture. In Solr 1.4 a replication master is a single node which receives all updates initially and keeps everything organized. Solr replication slave nodes receive no updates directly, instead all changes (such as inserts, updates, deletes, etc.) are made against the single master node. Changes made on the master are distributed to all the slave nodes, which service all query requests from the clients:

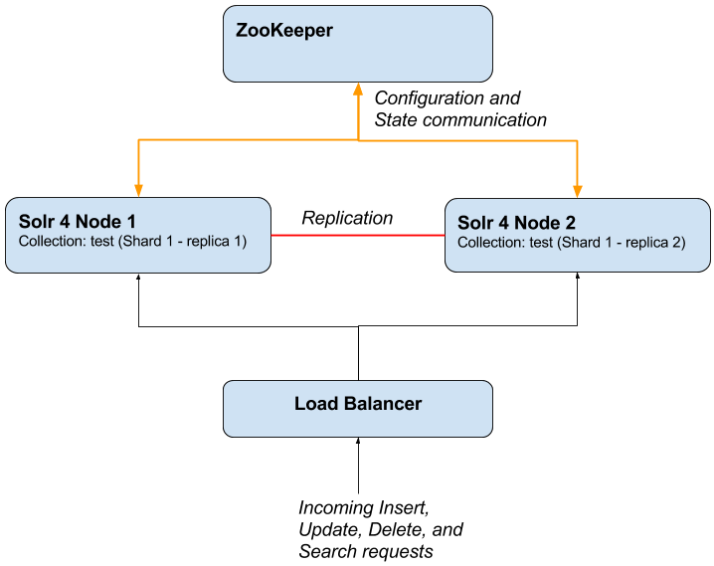


All of the changes to the index are written to the Master server. The Slaves are read-only copies of the Master, and services all search requests. The index is replicated by copying files from the Master server to the Slave servers. The Slave servers poll the Master server for updates, and when there are changes to the index the Slave servers will download the changes via HTTP.

There are a couple of problems that we need to be aware of with this replication setup:

- If the Master goes down no updates are possible and we have to manually promote a Slave to become the new Master. Search still work though, which is the important part in most solutions.
- When the index is rather small, let's say 1-2 GB, there is no real problem of firing up new slave servers or replicating updates to existing servers. The copying of the index to the new node will go quite quickly (at least 10mb per second). However, when the index grows and we reach tens of GBs, then copying the index from the Master to a new Slave node may become a problem. Let's say the index is 50GB, then it would take you around 1 ½ hour with 10mb per second speed. And if the mergeFactor is set really low, then updates to indexes will replicate very slowly to the slaves.

In Solr 4.0 with a SolrCloud cluster there is no explicit concept of Master/Slave nodes. Instead each node is a fully independent peer node that gets its configuration from ZooKeeper and updates its state in ZooKeeper. Indexing requests are replicated from one node to the other:



To create a functioning SolrCloud, you'll need at least 2 running instances of Solr and a Zookeeper to manage them. These Solr nodes would contain an index with a single shard, replicated across both nodes. However, this configuration creates a single point of failure, the Zookeeper, and therefore is not ideal for a truly fault tolerant architecture. Instead collection of Zookeepers should be used, referred to as a Zookeeper Ensemble.

Snapshot

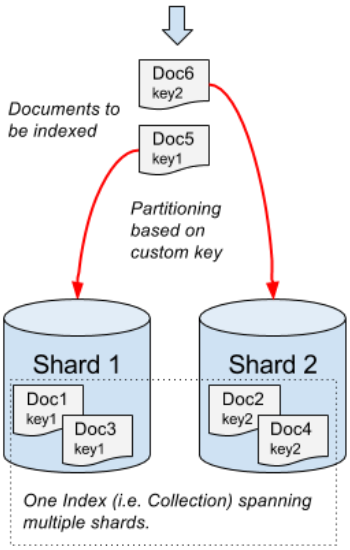
This is a directory containing hard links to the data files of an index. Snapshots are distributed from the Master nodes when the Slaves pull them, "smart copying" any segments the Slave node does not have in the snapshot directory that contains the hard links to the most recent index data files.

Sharding

This section goes into the sharding concept.

Shard

In the Solr world you sometimes get indexes growing too big to manage. In these cases the index, or collection as it is referred to, can be partitioned into smaller pieces where each partition holds part of the index. These partitions are also referred to as shards. What documents goes into what shard then? Solr doesn't have any logic for distributing documents (i.e. indexed data) over shards. The partition key is selected by the solution implementer, could be a computed hash, created date, name, or whatever other property make sense to get an even partitioning. We will see in the Alfresco - Solr integrations section what partition key Alfresco uses.
The following picture illustrates partitioning of documents into different shards:



In the above picture shard 1 and shard 2 can be on different nodes in the cluster (most common) or on the same node.
The document ID is unique across all shards as it needs to be unique for a whole index.
The reason why you need sharding can be summarized as follows:

- Data size grows beyond RAM on a single box
  - Lucene can handle this, but there's a performance cost
- Data size grows beyond local disk
- Latency requirements - response times are not good enough
  - Garbage Collections blocking queries
- Might not be trivial to open up and analyze a 32GB heap dump on your Laptop (or bigger depending on how much RAM is needed)

Having sharding obviously introduces new problems:

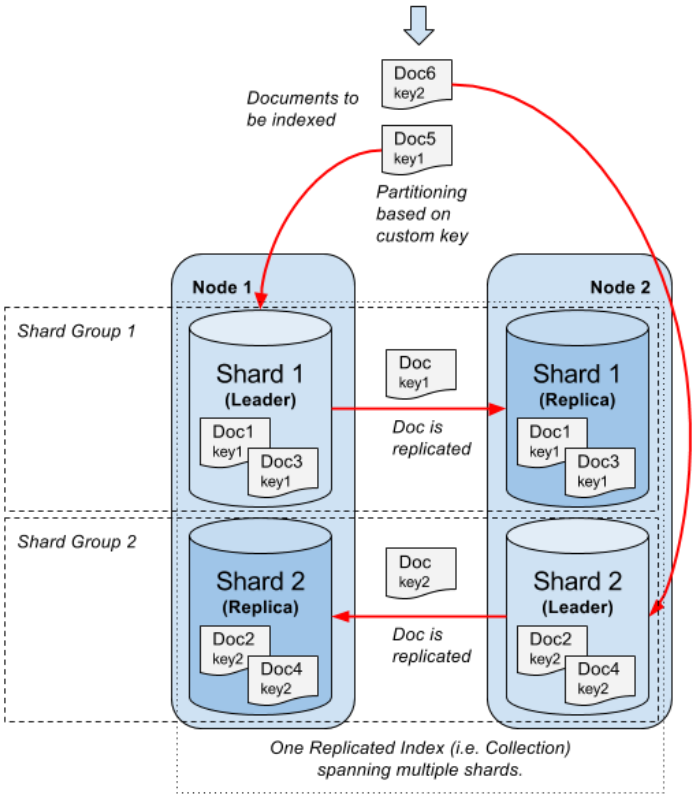
- Partial failures
- Lagging shards
- Synchronizing cluster state and configuration
- Network partitions
- Distributed IDF issues - now varies across shards, biasing ranking

Inverse Document Frequency (IDF)

Term Frequency and Inverse Document Frequency are used when calculating the relevancy for a document in the search result. Term frequency counts the number of times a term appears within the field we are querying in the current document. The more times it appears, the more relevant the document is. The Inverse Document Frequency takes into account how often a term appears as a percentage of all the documents in the index. The more frequently the term appears, the less weight it has.
For performance reasons, Solr 4 doesn't calculate the IDF across all documents in the collection (index). Instead, each shard calculates a local IDF for the documents contained in that shard.
Calculating the IDF value present no problem as long as we got only one shard. However, if there are multiple shards and the collection (index) is heavily skewed in its distribution across shards, then you may find misleading relevancy results in your searches.
In practice, this is not a problem. The differences between local and global IDF diminish the more documents that you add to the collection (index). With real-world volumes of data, the local IDFs soon even out. The problem is not that relevance is broken but situations when there is too little data.
This is important to know during testing, when you might not have access to relevant data volume. So search results might sometimes not appear correct in what documents are displayed as most relevant.

Leader, Replica, and Shard Group

Each shard can be a leader or a replica. The leader shard takes the incoming indexing requests and then replicates them over to the replica (note that there can be more than one replica):



It is common to have the replicas spread out on different cluster nodes for improved fault tolerance. The shard leader and its replicas form what is called a shard group. The shard leader increments the `_version_` field on the new or updated document.
This setup serves two purposes:

- **High availability** - queries can be distributed both to the leader and the replicas (the leader is essentially also a replica)
- **Fault tolerance** - if a leader goes down one of the replicas take over as new leader

Searching across Shards (Distributed Search)

The ability to search across shards is built into the Solr query request handlers. You do not need to do any special configuration to activate it. In order to search across two shards, you would issue a search request to Solr, and specify in a `numShards` URL parameter a comma delimited list of all of the shards to distribute the search across. You can issue the search request to any Solr instance, and the server will in turn delegate the same request to each of the Solr servers identified in the `numShards` parameter. The server will aggregate the results and return the standard response format.

Collection

A collection is a essentially a single index (i.e. core) that spans many shards. It is defined by a named configuration stored in ZooKeeper, the number of shards, document routing strategy, and replication factor (how many copies of each document in the collection).
There is a REST API available to work with collections, to create a new collection you would use the following type of REST API call:

```
$ curl "http://localhost:8983/solr/admin/collections?
action=CREATE&name=website&replicationFactor=2&numShards=2&collection.configName=website
```

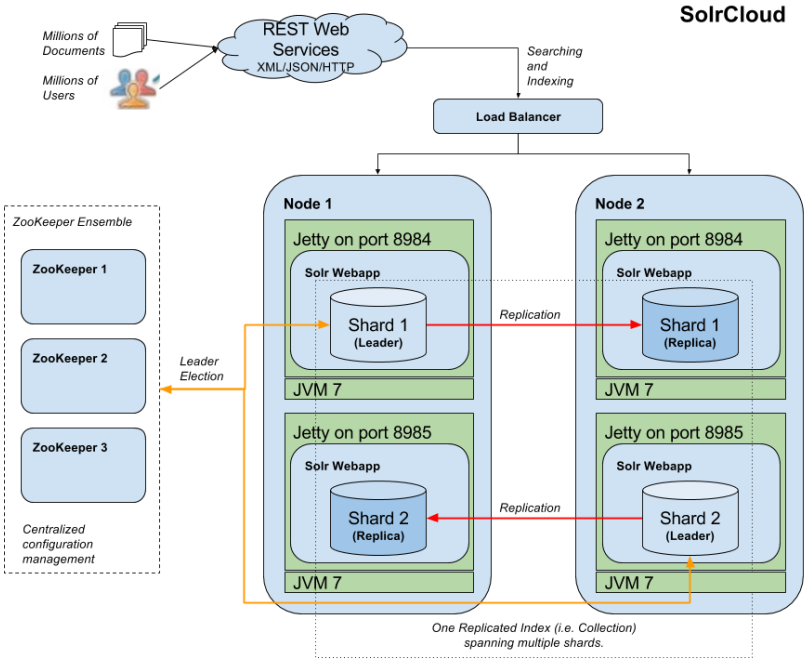
In this case I am telling Solr to create a new index for indexing website pages. There should be 2 shards splitting the index in 2. Each shard should have one replica (i.e. leader (also a replica) + replica = replication factor 2). The `webiste` config should already be present in ZooKeeper.

SolrCloud

(*Important, SolrCloud is not used by Alfresco and I only include it here for reference, if you are not interested in it, then skip to next section*)
The sharding and distributed search features in Solr version 4 are referred to as SolrCloud. Passing parameters to enable these capabilities will enable you to set up a highly available, fault tolerant cluster of Solr servers. Use SolrCloud when you want high scale, fault tolerant, distributed indexing and search capabilities.
Before Solr 4.0, and SolrCloud, you would have to take care of a number of things yourself if you wanted to setup Solr in a cluster, such as data distribution, setting up replication, thinking about data duplication, and so on.

ZooKeeper

In order to run SolrCloud you need to have Apache ZooKeeper installed. Zookeeper is a centralized service for maintaining configurations, naming, and provisioning service synchronization. SolrCloud uses ZooKeeper to synchronize configuration and cluster states (such as elected shard leaders):



ZooKeeper is used as a repository for cluster configuration and coordination – think of it as a distributed file system that contains information about all of the Solr servers. A single ZooKeeper service can handle the cluster but then it becomes a single point of failure. In order to avoid such a scenario, it is recommended that a ZooKeeper Ensemble, i.e. running multiple ZooKeeper servers in concert, is in action.
Every ZooKeeper server needs to know about every other ZooKeeper server in the ensemble, and a majority of servers (called a Quorum) are needed to provide service. For example, a ZooKeeper ensemble of 3 servers allows anyone to fail with the remaining 2 constituting a majority to continue providing service. If you have 5 ZooKeeper servers then up to 2 servers can fail at a time, and so on.
Here we can also see that each shard (leader replica and the other replicas) are run inside the usual Solr webapp. Search requests comes in via a load balancer and are distributed across the shards specified in the URL parameter `numShards`. As the search request load increases the load can be distributed across the shard replicas.

Transaction Log

SolrCloud needs a transaction log in order to operate properly. The transaction log keeps track of all the uncommitted changes to the index. It is an append-only log of write operations maintained by each node. It records all write operations performed on an index between two commits. Anytime the indexing process is interrupted, any uncommitted updates can be replayed from the transaction log.

Overseer

The overseer is a special node that executes cluster administration commands and writes updated state to ZooKeeper. Handles automatic fail-over and leader election.

Hands-on Solr

Let's play around with the standard Solr features before we dig into the Alfresco-Solr integration, so we can compare when looking at how Alfresco uses Solr.

Solr Master-Slave Replication

Let's try out the traditional Solr Master-Slave setup. This requires at least two servers, one for the Master and one for the Slave. We can easily set this up on a single box as follows. For more detailed information about Solr Replication see the online docs.

Install Solr

Start by installing Solr. On my Ubuntu box I installed it manually like this (Java is assumed to be already installed):

```
$ wget http://archive.apache.org/dist/lucene/solr/4.10.3/solr-4.10.3.tgz
$ tar -xvf solr-4.10.3.tgz
$ mkdir solr
$ cp -R solr-4.10.3/example solr
```

Since we'll need two Solr servers for this example, simply make 2 copies of the **example** directory -- making sure you don't have any data already indexed:

```
Temp/solr$ rm -r example/solr/collection1/data/*
Temp/solr$ cp -r example master
Temp/solr$ cp -r example slave
```

Setting up the Solr Master

A small configuration change is needed before we can start the Master node. Open up the **solrconfig.xml** file located in the **.../master/solr/collection1/conf** directory and change the `/replication` handler configuration as follows:

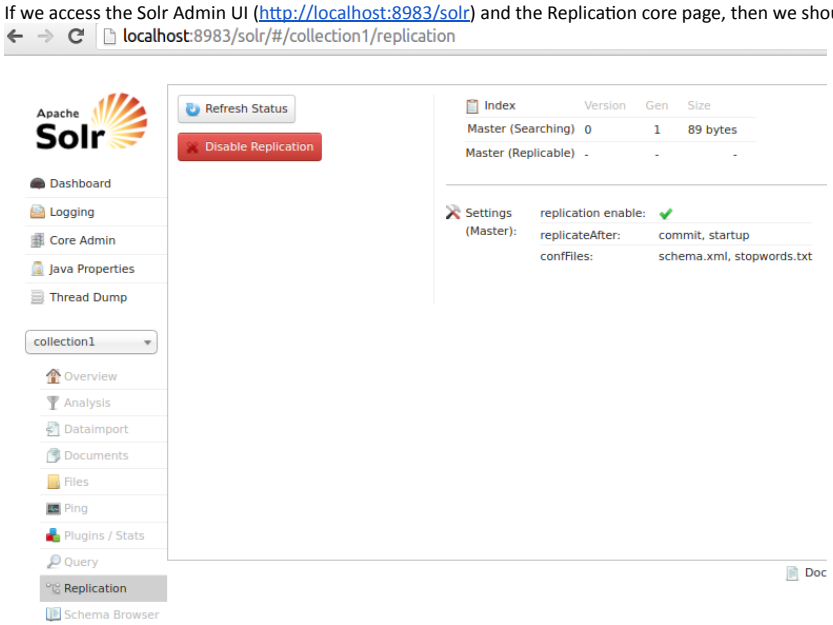
```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
    <lst name="master">
        <str name="replicateAfter">commit</str>
        <str name="replicateAfter">startup</str>
```

```
        <str name="confFiles">schema.xml,stopwords.txt</str>
    </lst>
</requestHandler>
```

To start the Master simply change into the newly created **master** directory and issue the following command:

```
Temp/solr$ cd master
Temp/solr/master$ java -jar start.jar
…
3022 [searcherExecutor-6-thread-1] INFO  org.apache.solr.core.SolrCore  – [collection1] Registered new searcher Searcher@22624190[collection1]
main{StandardDirectoryReader(segments_1:1:nrt)}
3025 [main] INFO  org.eclipse.jetty.server.AbstractConnector  – Started SocketConnector@0.0.0.0:8983
```

If we access the Solr Admin UI (http://localhost:8983/solr) and the Replication core page, then we should see something like this:



The `replicateAfter` and `confFiles` values should match what we configured in **solrconfig.xml**.
So, starting a Master Solr server is easy, just a small configuration change and then start it up.

Setting up the Solr Slave

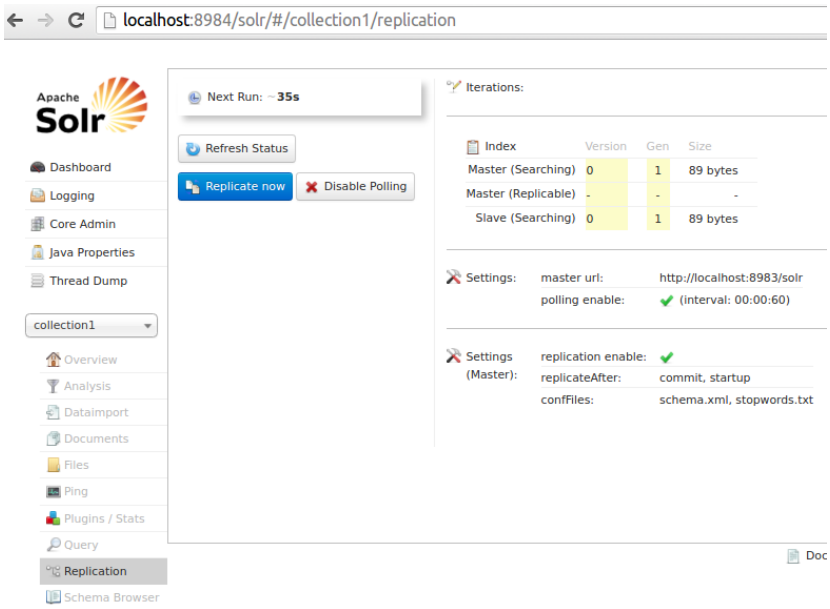
Now, let's get the Slave node going. This will require a little configuration change to tell the Slave where the Master node is running so it can poll for updates over HTTP:
Open up the **solrconfig.xml** file located in the **.../slave/solr/collection1/conf** directory and change the `/replication` handler configuration so it points to the Master:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
    <lst name="slave">
      <str name="masterUrl">http://localhost:8983/solr</str>
      <str name="pollInterval">00:00:60</str>
    </lst>
</requestHandler>
```

Then start the Solr node on a different port:

```
Temp/solr$ cd slave
Temp/solr/slave$ java -Djetty.port=8984 -jar start.jar
```

If we access the Solr Admin UI (http://localhost:8984/solr) and the Replication core page for the Slave node, then we should see something like this:



Here we can see the **solrconfig.xml** configuration for both Master and Slave.

Indexing some documents via the Master

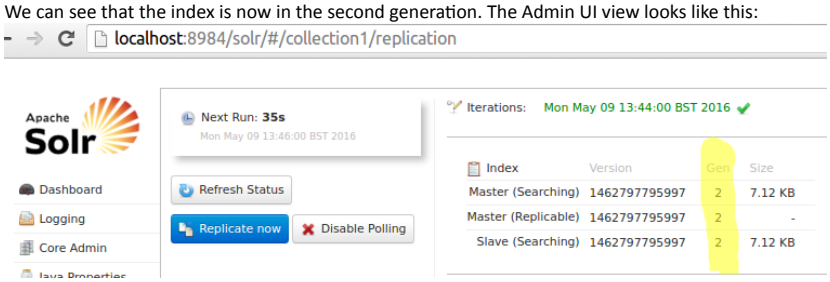
Step into the **exampledocs** directory in the **master** directory structure and index a few documents:

```
martin@gravitonian:~$ cd Temp/solr/master/exampledocs/
martin@gravitonian:~/Temp/solr/master/exampledocs$ java -Durl=http://localhost:8983/solr/collection1/update -jar post.jar ipod_video.xml
SimplePostTool version 1.5
Posting files to base url http://localhost:8983/solr/collection1/update using content-type application/xml..
POSTing file ipod_video.xml
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/collection1/update..
Time spent: 0:00:00.283
```

We should see calls to both the /update handler and the /replication handler in the Master logs:

```
430603 [qtp1778535015-11] INFO  org.apache.solr.update.processor.LogUpdateProcessor  – [collection1] webapp=/solr path=/update params={commit=true}
{commit=} 0 139
474472 [qtp1778535015-11] INFO  org.apache.solr.core.SolrCore  – [collection1] webapp=/solr path=/replication params=
{qt=/replication&wt=javabin&version=2&command=indexversion} status=0 QTime=1
```

We can see that the index is now in the second generation. The Admin UI view looks like this:



Searching via Slave(s)

Search for the document on the Slave with http://localhost:**8984**/solr/collection1/select?q=*:*

Playing around with SolrCloud

(*Important, SolrCloud is not used by Alfresco and I only include it here for reference, if you are not interested in it, then skip to next section*)
Now on to SolrCloud, which is a bit different then the Master-Slave setup. Let's play around with a standard SolrCloud installation to get a feel for it. It is quite easy to install SolrCloud.

Install Apache ZooKeeper

First we need to set up Apache ZooKeeper. This can be done in different ways depending on platform. Since the ZooKeeper package is available in Ubuntu's default repositories, I installed it using `apt-get`:

```
$ sudo apt-get install zookeeperd
```

After the installation completes, ZooKeeper will be started as a daemon automatically. By default, it will listen on port 2181.
To make sure that it is working, connect to it via Telnet:

```
$ telnet localhost 2181
Trying 127.0.0.1...
Connected to localhost.
```

```
Escape character is '^]'.
ruok
imokConnection closed by foreign host.
```

At the Telnet prompt, type in `ruok` and press ENTER.
If everything's fine, ZooKeeper will say `imok` and end the Telnet session.

Install Solr

Then install Solr, unless you already got it from looking at the replication feature. On my Ubuntu box I installed it manually like this (Java is assumed to be already installed):

```
$ wget http://archive.apache.org/dist/lucene/solr/4.10.3/solr-4.10.3.tgz
$ tar -xvf solr-4.10.3.tgz
$ mkdir solr
$ cp -R solr-4.10.3/example solr
```

Setting up a 2 shard cluster

First example we are going to play with is a 2 shard cluster on the same machine:

This example simply creates a cluster consisting of two Solr servers representing two different shards of a collection. Since we'll need two Solr servers for this example, simply make a copy of the example directory for the second server -- making sure you don't have any data already indexed:

```
Temp/solr$ rm -r example/solr/collection1/data/*
Temp/solr$ cp -r example example2
```

Now, start the first Solr server and populate ZooKeeper with our core (index) configuration and cluster configuration. We will set up a 2 shards cluster:

```
Temp/solr$ cd example
Temp/solr/example$ java -Dbootstrap_confdir=./solr/collection1/conf -Dcollection.configName=local2shards -DnumShards=2 -DzkHost=localhost:2181 -jar
start.jar
```

The parameters have the following meaning:

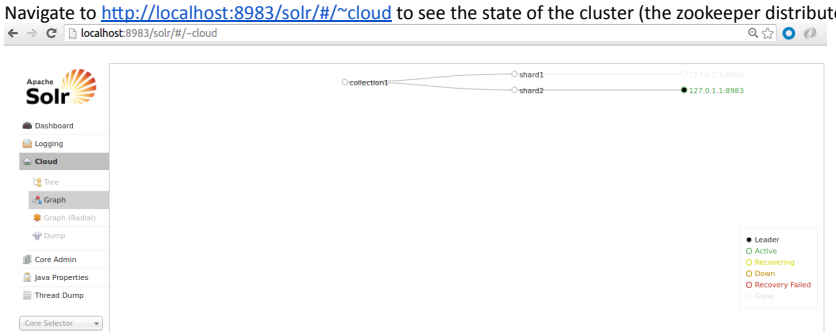
- `bootstrap_confdir`: since we don't yet have a config in zookeeper, this parameter causes the local configuration directory `./solr/collection1/conf` to be uploaded as the `local2shards` config.
- `collection.configName`: sets the config to use for the new collection. Omitting this param will cause the config name to default to `configuration1`.
- `numShards`: the number of logical partitions we plan on splitting the index into.
- `zkHost` - where is ZooKeeper running

The logs should show connection to ZooKeeper, config upload, shard start:

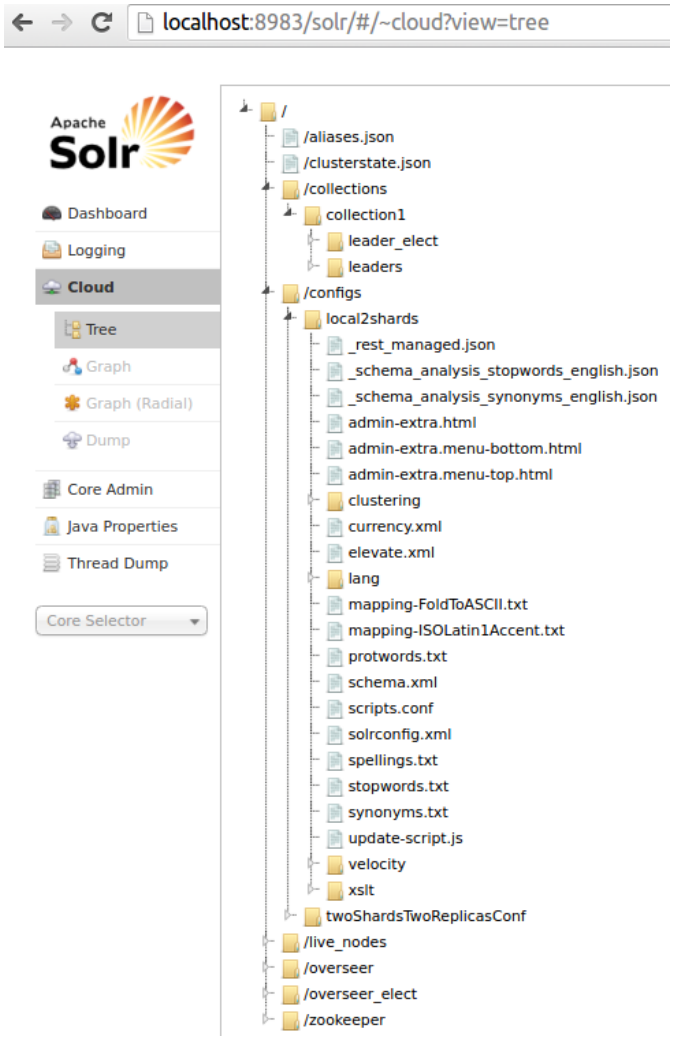
```
1538 [main] INFO  org.apache.solr.core.ZkContainer  – Zookeeper client=localhost:2181
1581 [main] INFO  org.apache.solr.common.cloud.ConnectionManager  – Waiting for client to connect to ZooKeeper
1609 [main-EventThread] INFO  org.apache.solr.common.cloud.ConnectionManager  – Watcher org.apache.solr.common.cloud.ConnectionManager@4b520ea8
name:ZooKeeperConnection Watcher:localhost:2181 got event WatchedEvent state:SyncConnected type:None path:null path:null type:None
1609 [main] INFO  org.apache.solr.common.cloud.ConnectionManager  – Client is connected to ZooKeeper
1670 [main] INFO  org.apache.solr.cloud.ZkController  – Register node as live in ZooKeeper:/live_nodes/127.0.1.1:8983_solr
1676 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /live_nodes/127.0.1.1:8983_solr
1686 [main] INFO  org.apache.solr.cloud.Overseer  – Overseer (id=null) closing
1697 [main] INFO  org.apache.solr.cloud.ElectionContext  – I am going to be the leader 127.0.1.1:8983_solr
1701 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /overseer_elect/leader
1706 [main] INFO  org.apache.solr.cloud.Overseer  – Overseer (id=95798155840126980-127.0.1.1:8983_solr-n_0000000004) starting
1767 [main] INFO  org.apache.solr.cloud.OverseerAutoReplicaFailoverThread  – Starting OverseerAutoReplicaFailoverThread
autoReplicaFailoverWorkLoopDelay=10000 autoReplicaFailoverWaitAfterExpiration=30000 autoReplicaFailoverBadNodeExpiration=60000
1797 [OverseerCollectionProcessor-95798155840126980-127.0.1.1:8983_solr-n_0000000004] INFO  org.apache.solr.cloud.OverseerCollectionProcessor  – Process
current queue of collection creations
1801 [main] INFO  org.apache.solr.common.cloud.ZkStateReader  – Updating cluster state from ZooKeeper...
1824 [OverseerStateUpdate-95798155840126980-127.0.1.1:8983_solr-n_0000000004] INFO  org.apache.solr.cloud.Overseer  – Replaying operations from work
queue.
1826 [OverseerStateUpdate-95798155840126980-127.0.1.1:8983_solr-n_0000000004] INFO  org.apache.solr.cloud.Overseer  – Update state numShards=2 message={
  "core":"collection1",
  "core_node_name":"core_node1",
  "roles":null,
  "base_url":"http://127.0.1.1:8983/solr",
  "node_name":"127.0.1.1:8983_solr",
  "numShards":"2",
  "state":"down",
  "shard":"shard2",
  "collection":"collection1",
  "operation":"state"}
1829 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/solrconfig.xml
1837 [zkCallback-2-thread-1] INFO  org.apache.solr.common.cloud.ZkStateReader  – A cluster state change: WatchedEvent state:SyncConnected
type:NodeDataChanged path:/clusterstate.json, has occurred - updating... (live nodes size: 1)
1853 [OverseerStateUpdate-95798155840126980-127.0.1.1:8983_solr-n_0000000004] INFO  org.apache.solr.cloud.Overseer  – Starting to work on the main queue
1854 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/elevate.xml
1869 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/mapping-FoldToASCII.txt
1878 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/admin-extra.menu-bottom.html
1884 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/lang/stopwords_da.txt
```

```
1898 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/lang/contractions_it.txt
...
2548 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/stopwords.txt
2555 [main] INFO  org.apache.solr.common.cloud.SolrZkClient  – makePath: /configs/local2shards/schema.xml
...
2641 [main] INFO  org.apache.solr.core.CoresLocator  – Looking for core definitions underneath /home/martin/Temp/solr/example/solr
2647 [main] INFO  org.apache.solr.core.CoresLocator  – Found core collection1 in /home/martin/Temp/solr/example/solr/collection1/
2648 [main] INFO  org.apache.solr.core.CoresLocator  – Found 1 core definitions
2650 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.cloud.ZkController  – publishing core=collection1 state=down collection=collection1
2650 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.cloud.ZkController  – numShards not found on descriptor - reading it from system property
2653 [zkCallback-2-thread-1] INFO  org.apache.solr.cloud.DistributedQueue  – LatchChildWatcher fired on path: /overseer/queue state: SyncConnected type
NodeChildrenChanged
2654 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.cloud.ZkController  – look for our core node name
2654 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.cloud.ZkController  – waiting to find shard id in clusterstate for collection1
2655 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.cloud.ZkController  – Check for collection zkNode:collection1
2655 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.cloud.ZkController  – Collection zkNode exists
2656 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.common.cloud.ZkStateReader  – Load collection config from:/collections/collection1
2657 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.common.cloud.ZkStateReader  – path=/collections/collection1 configName=twoShardsTwoReplicasConf
specified config exists in ZooKeeper
...
3655 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.core.SolrCore  – New index directory detected: old=null
new=/home/martin/Temp/solr/example/solr/collection1/data/index/
3656 [coreLoadExecutor-6-thread-1] WARN  org.apache.solr.core.SolrCore  – [collection1] Solr index directory '/home/martin/Temp/solr/example/solr/collection1/data/index' doesn't
exist. Creating new index...
3664 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.core.CachingDirectoryFactory  – return new directory for
/home/martin/Temp/solr/example/solr/collection1/data/index
3708 [coreLoadExecutor-6-thread-1] INFO  org.apache.solr.core.SolrCore  – SolrDeletionPolicy.onCommit: commits: num=1
...
3984 [OverseerStateUpdate-95798155840126980-127.0.1.1:8983_solr-n_0000000004] INFO  org.apache.solr.cloud.Overseer  – Update state numShards=2 message={
  "core":"collection1",
  "core_node_name":"core_node1",
  "roles":null,
  "base_url":"http://127.0.1.1:8983/solr",
  "node_name":"127.0.1.1:8983_solr",
  "numShards":"2",
  "state":"active",
  "shard":"shard2",
  "collection":"collection1",
  "operation":"state"}
4096 [zkCallback-2-thread-1] INFO  org.apache.solr.common.cloud.ZkStateReader  – A cluster state change: WatchedEvent state:SyncConnected
type:NodeDataChanged path:/clusterstate.json, has occurred - updating... (live nodes size: 1)
```

Navigate to http://localhost:8983/solr/#/~cloud to see the state of the cluster (the zookeeper distributed filesystem):



Navigating to **Cloud/Tree** displays ZooKeeper filesystem with the configurations and cluster state:

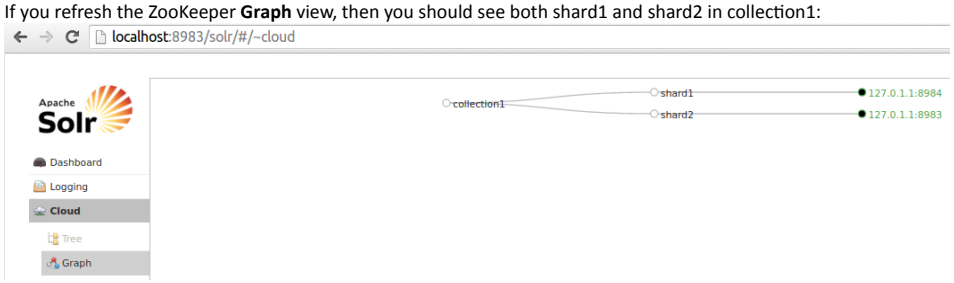


You can see from the ZooKeeper browser that the Solr configuration files were uploaded under **local2shards**, and that a new document collection named **collection1** was created. Under collection1 is a list of shards, the pieces that make up the complete collection.
Configuration is now populated in ZooKeeper and we are ready to kick off the second shard:

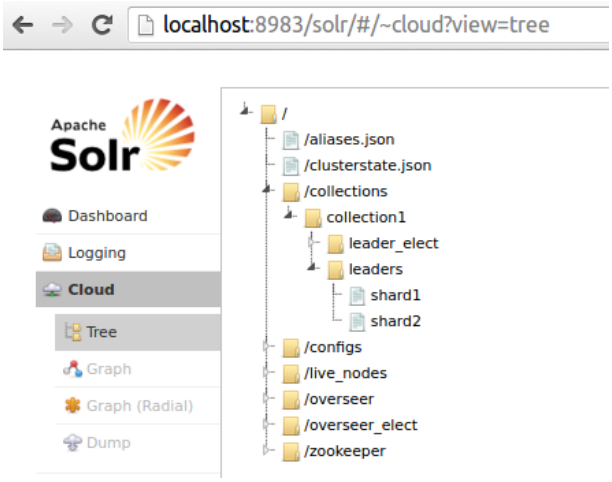
```
Temp/solr$ cd example2
Temp/solr/example2$ java -DzkHost=localhost:2181 -Djetty.port=8984 -jar start.jar

3935 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.ZkController  - Register replica - core:collection1 address:http://127.0.1.1:8984/solr
collection:collection1 shard:shard1
3936 [main] INFO  org.apache.solr.servlet.SolrDispatchFilter  - user.dir=/home/martin/Temp/solr/example2
3937 [main] INFO  org.apache.solr.servlet.SolrDispatchFilter  - SolrDispatchFilter.init() done
3944 [coreZkRegister-1-thread-1] INFO  org.apache.solr.common.cloud.SolrZkClient  - makePath: /collections/collection1/leader_elect/shard1/election
3965 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.ShardLeaderElectionContext  - Running the leader process for shard shard1
3972 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.ShardLeaderElectionContext  - Enough replicas found to continue.
3972 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.ShardLeaderElectionContext  - I may be the new leader - try and sync
3973 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.SyncStrategy  - Sync replicas to http://127.0.1.1:8984/solr/collection1/
3973 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.SyncStrategy  - Sync Success - now sync replicas to me
3974 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.SyncStrategy  - http://127.0.1.1:8984/solr/collection1/ has no replicas
3974 [coreZkRegister-1-thread-1] INFO  org.apache.solr.cloud.ShardLeaderElectionContext  - I am the new leader: http://127.0.1.1:8984/solr/collection1/
shard1
3976 [coreZkRegister-1-thread-1] INFO  org.apache.solr.common.cloud.SolrZkClient  - makePath: /collections/collection1/leaders/shard1
3976 [main] INFO  org.eclipse.jetty.server.AbstractConnector  - Started SocketConnector@0.0.0.0:8984
3987 [searcherExecutor-6-thread-1] INFO  org.apache.solr.core.SolrCore  - [collection1] webapp=null path=null params=
{q=static+firstSearcher+warming+in+solrconfig.xml&distrib=false&event=firstSearcher} hits=0 status=0 QTime=54
3987 [searcherExecutor-6-thread-1] INFO  org.apache.solr.core.SolrCore  - QuerySenderListener done.
3988 [searcherExecutor-6-thread-1] INFO  org.apache.solr.handler.component.SpellCheckComponent  - Loading spell index for spellchecker: default
3988 [searcherExecutor-6-thread-1] INFO  org.apache.solr.handler.component.SpellCheckComponent  - Loading spell index for spellchecker: wordbreak
3989 [searcherExecutor-6-thread-1] INFO  org.apache.solr.core.SolrCore  - [collection1] Registered new searcher Searcher@153f8a11[collection1]
main{StandardDirectoryReader(segments_1:1:nrt)}
```

If you refresh the ZooKeeper **Graph** view, then you should see both shard1 and shard2 in collection1:



And refreshing the **Tree** view shows now two leader shards (we got no replicas yet):



Index and Search some Documents

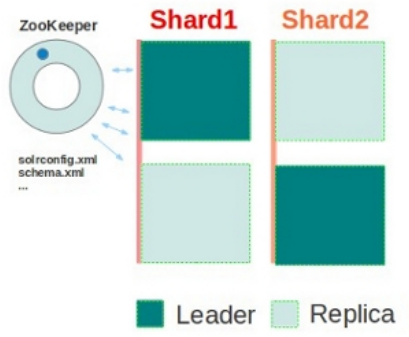
Randomly choose which shard instance to add documents too - they will be automatically forwarded to where they belong:

```
martin@gravitonian:~/Temp/solr/example$ cd exampledocs/
martin@gravitonian:~/Temp/solr/example/exampledocs$ java -Durl=http://localhost:8984/solr/collection1/update -jar post.jar ipod_video.xml
SimplePostTool version 1.5
Posting files to base url http://localhost:8984/solr/collection1/update using content-type application/xml..
POSTing file ipod_video.xml
1 files indexed.
COMMITting Solr index changes to http://localhost:8984/solr/collection1/update..
Time spent: 0:00:00.427
martin@gravitonian:~/Temp/solr/example/exampledocs$ java -Durl=http://localhost:8983/solr/collection1/update -jar post.jar monitor.xml
SimplePostTool version 1.5
Posting files to base url http://localhost:8983/solr/collection1/update using content-type application/xml..
POSTing file monitor.xml
1 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/collection1/update..
Time spent: 0:00:00.281
martin@gravitonian:~/Temp/solr/example/exampledocs$ java -Durl=http://localhost:8984/solr/collection1/update -jar post.jar mem.xml
SimplePostTool version 1.5
Posting files to base url http://localhost:8984/solr/collection1/update using content-type application/xml..
POSTing file mem.xml
1 files indexed.
COMMITting Solr index changes to http://localhost:8984/solr/collection1/update..
Time spent: 0:00:00.134
```

Search for these document with http://localhost:8983/solr/collection1/select?q=*:*
A request to either server results in a distributed search that covers the entire collection.

Adding replicas to the 2 shard cluster

This example will simply build off of the previous example by creating another copy of shard1 and shard2. Extra shard copies (i.e. replicas) can be used for high availability and fault tolerance, or simply for increasing the query capacity of the cluster:



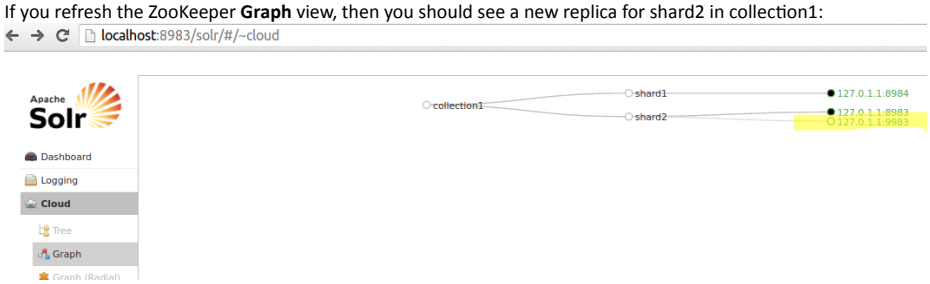
First, make sure you have run through the previous example so there are two shards and some documents indexed into each. Then simply make a copy of those two servers:

```
martin@gravitonian:~/Temp/solr$ cp -r example exampleB
martin@gravitonian:~/Temp/solr$ cp -r example2 example2B
martin@gravitonian:~/Temp/solr$ ls -l
total 16
drwxr-xr-x 15 martin martin 4096 Mar  9 13:20 example
drwxr-xr-x 15 martin martin 4096 Apr 27 15:43 example2
drwxr-xr-x 15 martin martin 4096 Apr 28 09:13 example2B
drwxr-xr-x 15 martin martin 4096 Apr 28 09:13 exampleB
```

Then start the two new servers (shard replicas) on different ports, each in its own window:

```
Temp/solr$ cd exampleB
Temp/solr/exampleB$ java -DzkHost=localhost:2181 -Djetty.port=9983 -jar start.jar
```

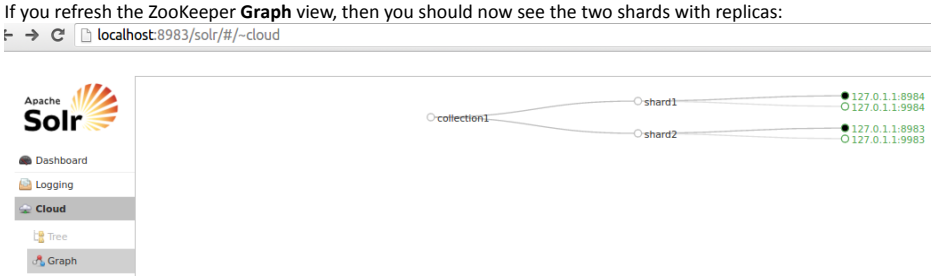
If you refresh the ZooKeeper **Graph** view, then you should see a new replica for shard2 in collection1:



Start the second replica:

```
Temp/solr$ cd example2B
Temp/solr/example2B$ java -DzkHost=localhost:2181 -Djetty.port=9984 -jar start.jar
```

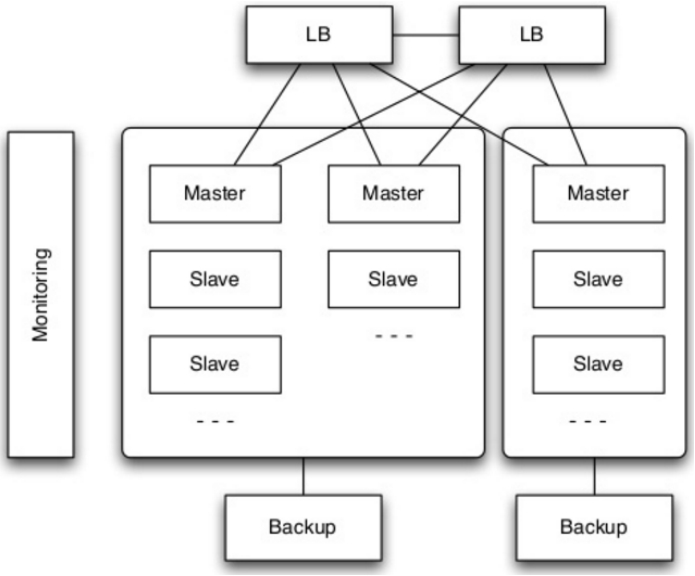
If you refresh the ZooKeeper **Graph** view, then you should now see the two shards with replicas:



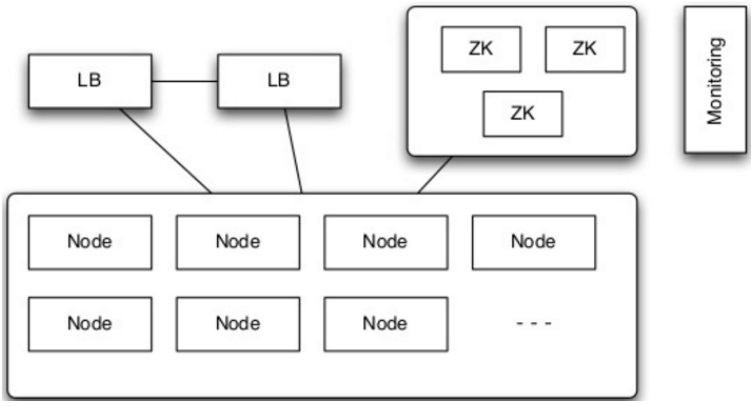
So as we can see, getting going with a little SolrCloud installation is quite easy. It gets trickier when leaders and replicas are on different hosts. Then you need to open up ports in the firewalls and make sure that the communication between the hosts are open.

SolrCloud Architecture

The SolrCloud architecture is based on the standard Solr server, so if you have not yet looked at the basic Solr Architecture see the first article.
If you have read the concepts section and played with SolrCloud you probably have a pretty good idea of the SolrCloud architecture. What might be interesting though is to compare pre version 4 scalable and fault tolerant architecture to version 4 scalable architecture.
Before version 4 you would use the traditional Master-Slave setup with load balancers:



With this architecture you would have to manage a lot in a custom way, such as routing. There would also be quite complex provisioning of nodes. Monitoring would have to be done on all the master and slave nodes.
With Solr version 4 and SolrCloud we got just nodes, automatic routing, and simple provisioning:



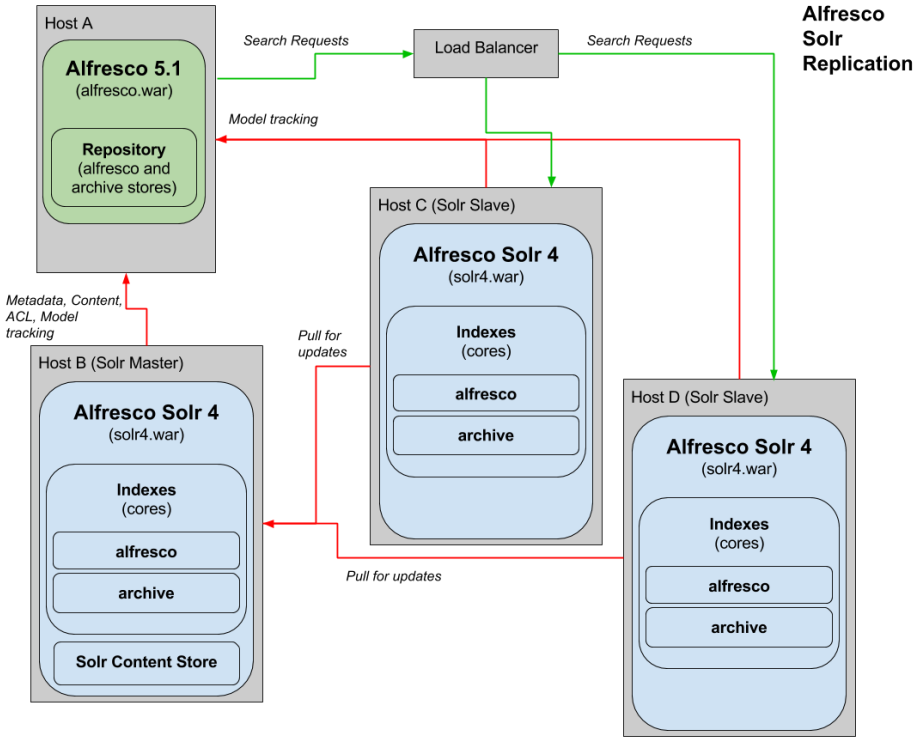
We also only need to monitor ZooKeeper as it monitors the cluster for us.

How is SolrCloud integrated with Alfresco?

Now the interesting part starts, how is SolrCloud used by Alfresco, *it is actually not*! You will not see any references to for example ZooKeeper and leader shards in the Alfresco documentation. Alfresco uses Solr in stand alone mode and has developed an entire sharding model that fits the ECM use case. The Alfresco Solr Replication solution also differs a bit from the standard Solr replication solution as the Alfresco Solr servers track the Alfresco server to update the index.

Replication

Setting up replication with Alfresco Solr works pretty much the same way as for the out-of-the-box Apache Solr. The Solr Master tracks Alfresco and Alfresco does searches via the Solr Slaves:



There are some things to be aware of. It is only the Solr Master server that does metadata, ACL, and content tracking against the Alfresco Repository. The Solr Slaves do only model tracking. This reduces the load on the Alfresco database and text transformation engine.
If the Master server goes down you would have to manually promote a Slave to become a new Master and then wait for it to get an up-to-date content cache. The Slaves don't do content tracking and therefore don't have a Solr Caching Content Store. However, a content cache miss is fine and the cache is just used for rebuilding the index, to cache content transformations, and to cache metadata.
Alfresco sends queries to the Slave we have set up in **alfresco-global.properties**. So we basically need a load balancer in front of the Slaves to distribute the load. No queries should be sent to the Solr Master server. Including the Master in queries will mean more variation in query results as it will be ahead of the Slaves. Including the Master in queries will slow down both indexing and query.
In the event of Master failure, Slaves can continue to service query requests. The most up to date Slave can be reconfigured as the new Master while the remaining Slaves process queries. Slaves can then be re-pointed (requires config change and restart) to the new Master in a rolling fashion in order to not take an outage.
To summarize, here are the advantages and disadvantages with Alfresco Solr Replication:

Advantages

- Splits read and write requests
- Load distribution for search
- High availability for search
- Any number of Slave instances can be created to scale search performance
- Usually less frequent index updates on the Slaves and better use of the cache

Disadvantages

- Increased latency (sum of tracking and Solr replication latency)
- Occasional large IO load to replicate large merges
- Complicated load balancing and management
- Reconfiguration if the Master is lost
- Cannot be used to scale the size of the index itself, see next section on sharding for that

Sharding

The most interesting part to be able to scale search is sharding. The documents stored in Alfresco may be divided up into a number of logical shards to support scaling the repository and the index. This division is at the moment only based on the Access Control Lists Identifier (ACL ID) of the documents. So documents with the same permissions will end up in the same shard. Documents live with their permissions, which means authorization evaluation is easily distributed.
A shard can have a number of "real" instances. A real instance of a shard is a single Solr core (which also represents a single lucene index). An instance is configured to pull information from the Alfresco Repository (i.e. Metadata tracking). As part of this pull the shard instance reports what information it contains. This is used to dynamically group shard instances together into shard groups. A group of similar shards represents a logical index that can be queried for all information from all the documents in Alfresco. An Alfresco shard group maps to a collection in Solr terminology.
When setting up sharding an Alfresco cluster is highly likely already in place as the use-case for deploying a sharding solution only comes into play when we got more than 50 million content items in the Repository.
Important:

- Sharding does not work with SSL
- Slave shards are not supported
- Solr backup is not used when index is sharded

Index Partitioning Key (Sharding Key)

Alfresco shards the index along the `ACL_ID`, which keeps both a repository node and its Access Control List (ACL) on the same Solr shard, speeding up permission checks as they can be done on the shard. Repository nodes can have their permissions changed, meaning the `ACL_ID` changes, and then move from one shard to the other.
Shards are numbered starting from zero. The shard identifier (i.e. shard number) is calculated as follows based on the ACL_ID and the number of shards the index should be split into:

```
shardId = ACL_ID % numShards
```

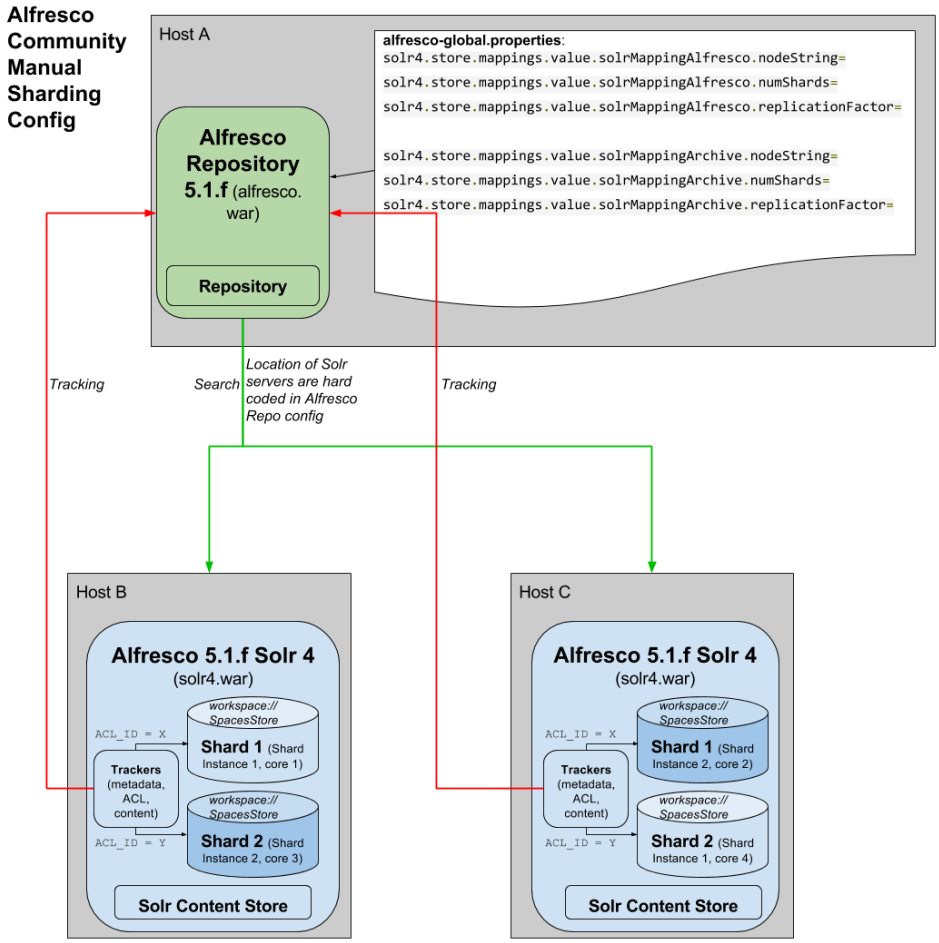
So if you know the `ACL_ID` you should be able to tell which server node the ACLs and related repository node go into based on the calculated `shardId`. Let's take the following example with a collection split up into 3 shards (0,1,2) and `ACL_ID` 1,3,4,8,11,33,230:

ACL_ID % numShards	shardId 0	shardId 1	shardId 2
1 % 3		1	
3 % 3	0		
4 % 3		1	
8 % 3			2
11 % 3			2
33 % 3	0		
230 % 3			2

So here we can quite quickly understand and see how the repository nodes will be distributed to different shards based on the `ACL_ID`.

Manual Sharding (Alfresco Community Edition)

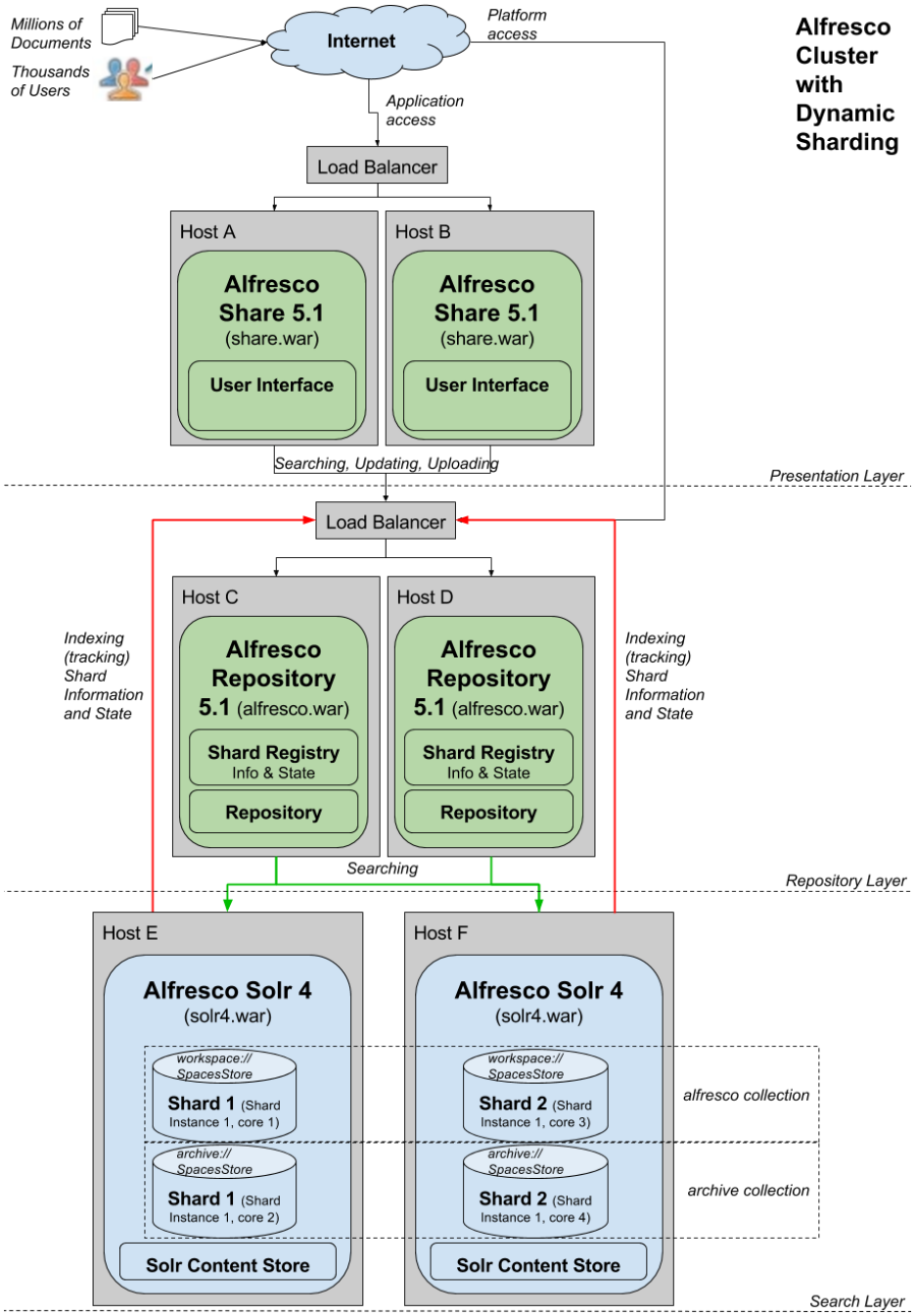
It is possible to configure and use sharding on the Alfresco Community edition. It might not be the most common use-case as you cannot use repository side clustering. Which means the use case has to be something like 50 million + nodes and very low load on the repository, so it is enough to just run with one Alfresco repository node, but you need index sharding because of the large amount of data (nodes).
We call this sharding setup **manual,** and the reason for this is because all configuration has to be done statically and manually with properties. There is no auto-discovery of shards via the so called Shard Registry.
The manual sharding solution looks something like this:



So in this Community setup we got Host A that contains the Alfresco Repository, which will be tracked by all the Solr nodes. The Alfresco Repository knows about the Solr nodes and shards via configuration in **alfresco-global.properties**.
Note that when running with this configuration the Alfresco Repository knows nothing about the shard states. The shard could be ok and up-to-date, lagging behind, or is simply unavailable. So the repo would just send a query to a shard without knowing if it is capable of executing it.

Dynamic Sharding (Alfresco Enterprise Edition)

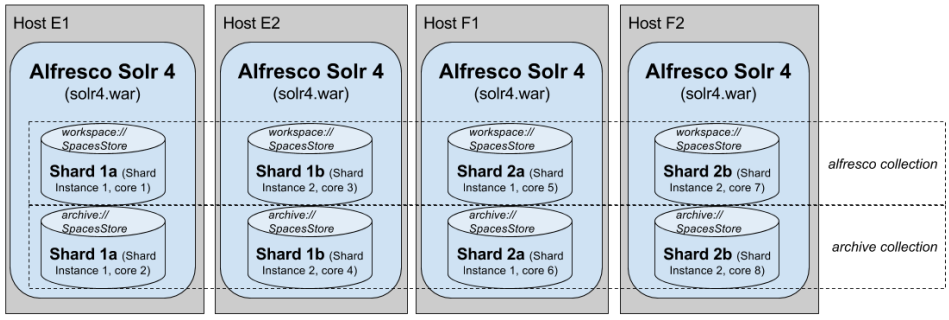
The Alfresco Enterprise edition has a lot of additional features compared to the Community edition. It is possible to cluster the Repository layer, auto-discover shards, distribute queries based on shard state etc.
The following picture shows a typical Alfresco cluster and how it uses sharded indexes:



You might be familiar with running a separate search layer before, maybe with replication or just a stand-alone Solr server. The difference here is that we got the index partitioned on multiple Solr servers. And each one of those need to track the Repository. The Solr servers can track any Repository node so going through a Load balancer makes sense for availability and fault tolerance. *Note however that the Solr Servers cannot be load balanced.* Each Repository server will call a Solr Server to execute a query based on the shard information and shard state that it keeps in its Shard Registry.

Having more than one instance of a shard adds resilience and **increases** the overall **query throughput**:

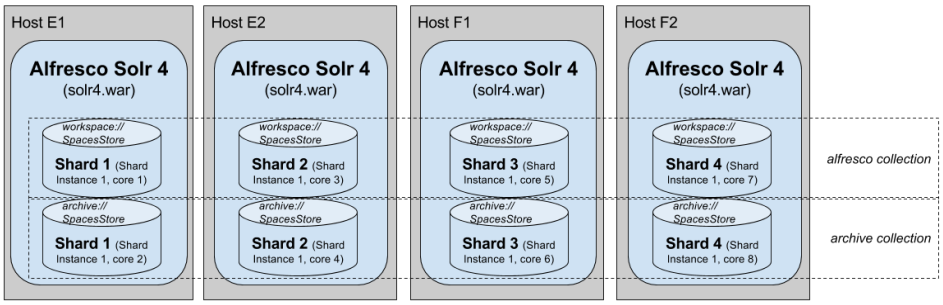
2 Shards
X
2 Shard Instances



Doubling the number of instances would roughly double the query throughput.

Doubling the number of shards would roughly **half the query time**:

4 Shards
X
1 Shard Instance



However, the query time is driven by the slowest shard. The more shards you have the more likely an instance will be slow compared with the rest. Slow shards may be caused by any number of reasons including: Garbage Collection (GC), hitting connection limits, uneven index distribution, query cache miss, filter cache miss, unusual facet distribution, etc. In general, query time scales linearly with index size.
Fault tolerance can be built into the sharding setup. For example, if we have a collection with X number of shards, and we want to be able to lose 2 Alfresco Solr nodes without loss of index data, then we would need to allocate at least 3 replicas per shard and have at least 3 Alfresco Solr nodes:

	Collection		
	Shard 1	**Shard 2**	**Shard 3**
Alfresco Solr Node 1	X	X	X
Alfresco Solr Node 2	X	X	X
Alfresco Solr Node 3	X	X	X

Dynamic sharding removes the requirement to follow a rigid shard pattern, as has to be done when using the Community version. Shards register with Alfresco as part of tracking and this information is kept in the Shard Registry. Dynamic shard registration is an enterprise feature. It is enabled by setting the following property in **alfresco-global.properties**:
`solr.useDynamicShardRegistration=true`
Each shard may have any number of replicas and the following properties govern which replicas are chosen for a query:

Property	**Description**	**Value**
`search.solrShardRegistry.purgeOnInit`	If `true` then remove persisted shard state from the database when the search subsystem starts.	`[true|false]`
`search.solrShardRegistry.shardInstanceTimeoutInSeconds`	If a shard has not made a tracking request within this time it will not be used for query.	`300`
`search.solrShardRegistry.maxAllowedReplicaTxCountDifference`	If any shard is more than this number of transactions behind the leading shard it will not be used.	`1000`

If there is more than one replica index for a repository store, then the most up to date will be used (the one that has indexes most transactions). For each shard, a replica is chosen at random from all the shards instances that are actively tracking.
The status of all available indexes, shards, replicas and other related information can be found via the Enterprise Admin Console or via JMX at "Alfresco - FlocAdmin".
Dynamic sharding does not require shards and replicas to be distributed correctly over a known set of hosts. Query is resilient, with a configurable delay, to replicas coming and going. For explicit sharding (i.e. manual sharding) all replicas must be available on the expected host at the expected URL. Dynamic shard registration allows different numbers of replicas for any shard: explicit sharding does not.
Shards are considered to be part of the same collection if they:

- Track the same Repository Store, such as `workspace://SpacesStore`
- Use the same core configuration template, such as `rerank` (and therefore SOLR schema etc)
- The number of shards in the collection is the same
- Use the same partitioning method with the same configuration if any is required
- Have the same setting to transform or ignore content.

All Alfresco Repository nodes that are used to provide information when building a shard replica should be part of the Alfresco cluster. Replica states are periodically stored using the Attribute Service and their clustered cache entries invalidated. Alfresco nodes need to be in the cluster to propagate replica state correctly. If this is not done then searches from non-clustered Repository nodes might use shards that are not available, is lagging behind etc. It is possible to reconfigure the related caches with more aggressive timeouts rather than having cluster based invalidation.
Each Alfresco Solr node maintains a Caching Content Store with the subset of nodes held in the shards it manages.

Shard Registry

This is a component that is part of the Alfresco Repository. Every Repository node in a cluster setup have this component. It knows all the shards and their states/configuration. It is used to select coordinating nodes and the nodes to make a full index. There can be multiple copies of an index with different shards numbers, templates, content etc. It works out which bits go together.
The Attribute Service is used to store the Shard Registry data, which needs clustering for all Alfresco instances to propagate updates. The Shard Registry is not exposed over Web Scripts, it is instead tracking that talks to it and query that uses it.
It is specific to the Alfresco Enterprise Edition.

Indexing

Each Alfresco Solr node tracks the Alfresco Repository. There are no master or slaves nodes, each node is functionally the same.

Searching

The Alfresco Repository will select a random Alfresco Solr node as query coordinator and tell it which other Solr nodes to use. Nodes are selected at random after excluding inactive shards or ones too far behind the one "in the lead with most data".
Remember: do not load balance access to Alfresco Solr nodes.

Alfresco Solr Core API

Alfresco Solr comes with a custom REST API for managing cores (i.e. shards).

Creating shards

The `newCore` action can be used to create new shards using an ACL based sharding key. The URL looks something like this:
http://localhost:8080/solr4/admin/cores?action=newCore&storeRef=workspace://SpacesStore&numShards=2&nodeInstance=1&replicationFactor=2&numNodes=2&template=rerank&property.alfresco.host=192.168.1.3
The different parameters have the following meaning:

Parameter	Description	Value
action	Specifies the action/operation that should be performed by the Solr server.	`[ newCore | updateCore | removeCore ]` (`newCore` is a custom Alfresco action to create new shard cores)
storeRef	Points to the Repository store that should be tracked (i.e. indexed) by the Solr server for this core.	`workspace://SpacesStore` (the store that has all the live content). Could also be `archive://SpacesStore` for soft deleted content.
numShards	The number of shards that makes up the index (i.e. collection).	`2` (The index will be split in half - 2 partitions)
nodeInstance	The node number where the shard is created.	`1` (The shard is created on the first node)
replicationFactor	Specifies how many replicas (shard instances) we want of the shard core.	`2` (There will be two identical copies of the shard core)
numNodes	How many nodes are the shards created on in total.	`2` (There are two hosts with Apache Tomcat running the Solr webapp)
template	Specifies the core configuration template to use for the new shard core.	`rerank` (The <solr home>/templates/rerank directory will be the base for the new shard core configuration. There are other templates but the rerank one is the recommended template to use.)
property.*	Can be used to set specific values for properties that should go into solrcore.properties for the core configuration.	`property.alfresco.host=192.168.1.3` (The `alfresco.host` property in the **solrcore.properties** file will be set to `192.168.1.3`. This file will be located in a directory such as /var/lib/tomcat7/data/rerank--workspace-SpacesStore--shards--2-x-2--node--1-of-2/workspace-SpacesStore-0/conf)

This REST call will create the appropriate shards on any given node from a specified node set if the following is true:
`(numShards * replicationFactor) % numNodes == 0`
So you need to have a number of nodes that allows for even distribution of the shards. You can for example have 3 shards and 3 replicas deployed on 3 nodes:
`(3 * 3) % 3 == 0`
Or you could have 3 shards and 3 replicas deployed on 9 nodes:
`(3 * 3) % 9 == 0`
But you could not have 3 shards and 3 replicas deployed on 8 nodes:
`(3 * 3) % 8 != 0`
When the `newCore` API is called the shards are created and distributed according to a predefined pattern based on this type of calculation:

```
var numNodes = 3;
var nodeInstance = 1;
var replicationFactor = 2;
var numShards = 3;
var test = 0;
var shardIds = "";
for (replica = 0; replica < replicationFactor; replica++) {
  for (shard = replica; shard < numShards + replica; shard++)  {
    if (test % numNodes == nodeInstance - 1) {
      var shardId = shard % numShards;
      shardIds += shardId + ",";
    }
    test++;
  }
}
document.getElementById('text').innerHTML = shardIds;
```

(You can run the above code in JSFiddle to see what `shardIds` you get for different combinations of parameter values)
If we have for example 3 shards with 2 replicas on 3 nodes then the shard distribution would look like this:

	shardId = 0	shardId = 1	shardId = 2
Node 1	X	X	
Node 2		X	X
Node 3	X		X

So as we can see the replicas for each shard are distributed out on different nodes for the purpose of redundancy and failover.
When creating shards for an Enterprise dynamic sharding solution it is also possible to specify what `shardIds` go into what node:
http://localhost:8080/solr4/admin/cores?action=newCore&storeRef=workspace://SpacesStore&numShards=5&nodeInstance=1&replicationFactor=2&numNodes=2&template=rerank&**shardIds=0,1,2,3,4**
In this case the above calculation is bypassed.

Updating shards

It is possible to update shard configuration with the `updateCore` action. You need to know the core name:
TODO

Removing shards

To remove a shard use the `removeCore` action. You need to know the core name:
http://localhost:8080/solr4/admin/cores?action=removeCore&&coreName=alfresco
This will remove the index directory, data directory, and the core instance directory. Note that the call is not shard aware, so you have to make this call on all Alfresco Solr Tomcats that contains shards that make up the collection.
It is also possible to use the `storeRef` instead of `coreName` (if the core name is constructed from the store reference):
http://localhost:8080/solr4/admin/cores?action=removeCore&storeRef=workspace://SpacesStore

Solr Core Configuration Templates

These are different Solr core (index) configurations that can be used to build shard instances. You can find the templates in the **<alfrescoinstalldir>/solr4/templates** directory. When you use the `newCore` Alfresco Solr Core API call the `template` parameter specifies from what subdirectory in the **/templates** directory to copy the core configuration from:
http://localhost:8080/solr4/admin/cores?action=newCore&storeRef=workspace://SpacesStore&numShards=2&nodeInstance=1&replicationFactor=2&numNodes=2&template=rerank
It is possible to create your own custom core configurations and put them in the /templates directory and use them for shard creation. The custom configurations can for example alter locale specific tokenisation, synonyms, stopwords, etc.
The new **rerank** template should always be used and it is a big improvement for query performance. The index is smaller and queries are faster for the same query functionality. Cross language search is still offered but it can be limited to just name. This alone cuts a big chunk off query complexity and repetition in the index. Rerank refers to 2 stage query processing - phrases are done as conjunctions and then reranked as phrases rather than run as phrases. This reduces phrase related IO (term positions).
For more information about the different templates that you can choose form see the following docs.

Hands-on Alfresco Solr

In this section we will get hands-on with the Alfresco Solr integration and set up a couple of different solutions.

Alfresco Solr Replication

In this section we will show an example of how to setup Solr replication with the Alfresco Community Edition (for more info see overview section).
The following nodes will participate in the setup:

- **Host 1**: with Alfresco Repository and Solr Master (Tomcat 10.244.61.136:8080)
- **Host 2**: with Solr Slave (Tomcat 10.244.61.62:8080)

Important: SSL is turned off between the repository server and the solr server(s).

Set up Alfresco Repository and Master Solr on Host 1 (10.244.61.136)

This includes installing Alfresco 5.1 Community and then configuring the Repository and Master Solr server.

Installing and Configuring Alfresco Repository

Download the latest Community Edition from https://www.alfresco.com/alfresco-community-download. Install Alfresco Community 5.1.f from the full installer, including Solr 4 (which we will use as Master Solr).
Don't start Alfresco at this point.
Now configure the Alfresco Repository Server to use only the Slave Solr server for query and turn off SSL. Open **alfresco-global.properties** set the following properties:

```
# Point searching to Slave (or slave load balancer IP)
solr.host=10.244.61.62
# Turn off https when replication is active
solr.secureComms=none
```

Configuring the Master Solr Server

A small configuration change is needed before we can start the Repository and the Solr Master node. Open up the **solrconfig.xml** file located in the **<alfrescoinstalldir>/solr4/workspace-SpacesStore/conf** directory and change the `/replication` handler configuration as follows:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
    <lst name="master">
        <str name="replicateAfter">commit</str>
        <str name="replicateAfter">startup</str>
        <str name="confFiles">schema.xml,stopwords.txt</str>
    </lst>
</requestHandler>
```

Now do the same change for the archived content configuration. Open up the **solrconfig.xml** file located in the **<alfrescoinstalldir>/solr4/archive-SpacesStore/conf** and make the change.
Next things we need to configure in each core is to use a plain connection when talking to the slave(s). Open up the **solrcore.properties** file located in the **<alfrescoinstalldir>/solr4/workspace-SpacesStore/conf** directory and update the following property:

```
alfresco.secureComms=none
```

Make the same change in the **solrcore.properties** file located in the **<alfrescoinstalldir>/solr4/archive-SpacesStore/conf** directory.
Alfresco also uses some other custom properties to control what Solr node is the Master and what Solr node(s) are the slaves. Again, open up the **solrcore.properties** file located in the **<alfrescoinstalldir>/solr4/workspace-SpacesStore/conf** directory and add the following properties:

```
enable.master=true
enable.slave=false
```

Make the same change in the **solrcore.properties** file located in the **<alfrescoinstalldir>/solr4/archive-SpacesStore/conf** directory.

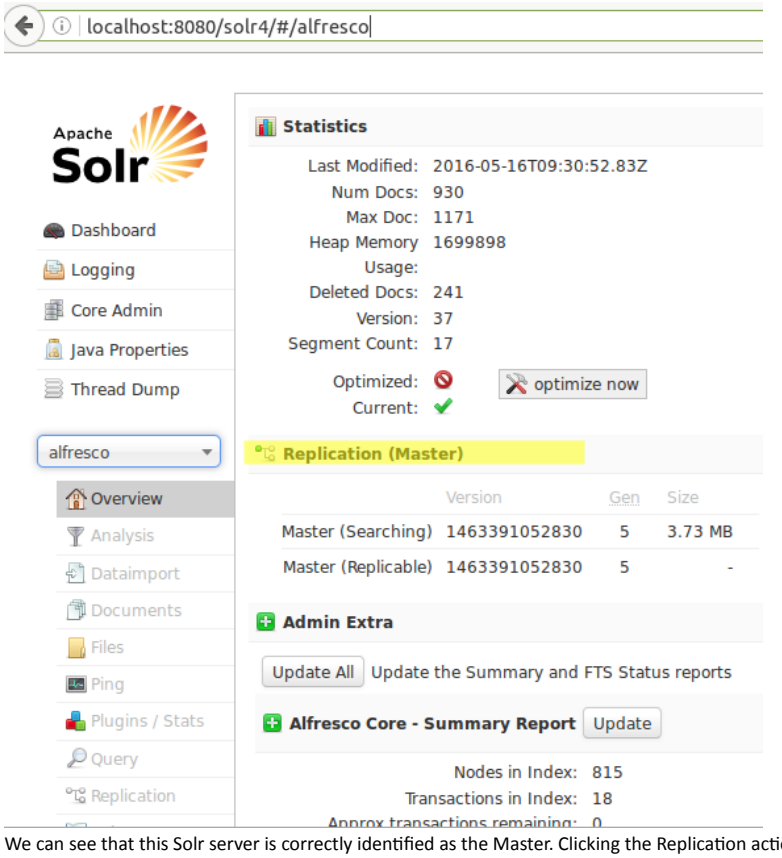
Start Alfresco Tomcat

After starting the Alfresco Tomcat server it should work to login to http://localhost:8080/share but searching will show connection failure in the log as the Solr Slave server is not yet up and running:

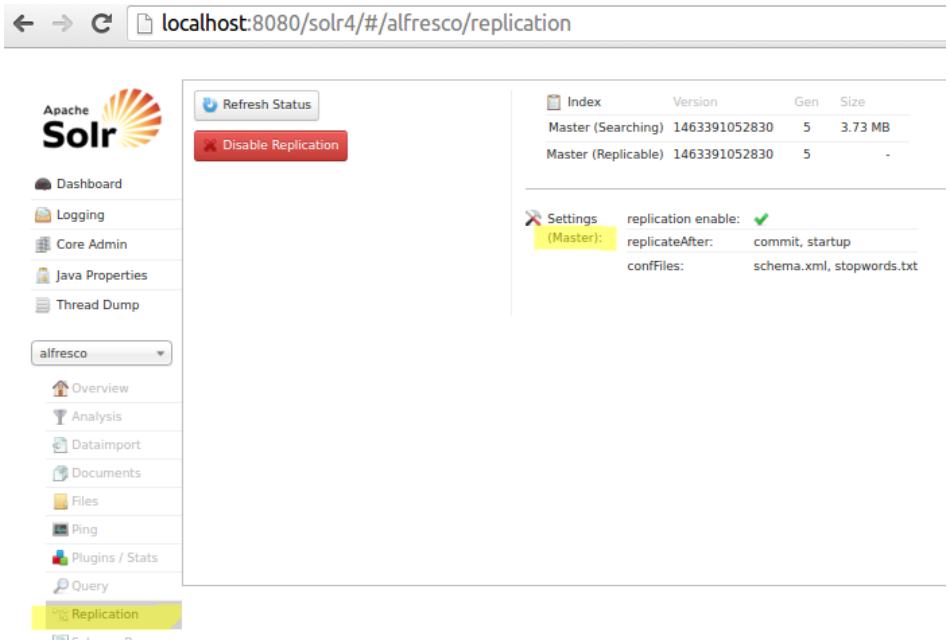
```
2016-05-16 10:30:58,434  ERROR [extensions.webscripts.AbstractRuntime] [http-apr-8080-exec-4] Exception from executeScript - redirecting to status
template error: 04160002 Wrapped Exception (with status template): 04160006 Failed to execute script
'classpath*:alfresco/templates/webscripts/org/alfresco/slingshot/documentlibrary/doclist.get.js': 04160005 Failed to execute search:  +@cm\:modified:
[2016\-5
```

Check the Solr Cores

We should now have the Solr Master server working for the two cores. So access the Solr Admin page to confirm. Open the http://localhost:8080/solr4/#/alfresco URL and look at the alfresco core:



We can see that this Solr server is correctly identified as the Master. Clicking the Replication action shows the config for the Master based on what was set in **solrconfig.xml**:



Set up the Solr Slave Server on Host 2 (10.244.61.62)

The first thing we need to do is download and install Apache Tomcat 7 on the Linux box:

```
$ sudo apt-get install tomcat7
```

This will install Tomcat and its dependencies, such as Java, and it will also create the tomcat7 user. It also starts Tomcat with its default settings.
Now start Tomcat 7 as follows:

```
$ sudo /etc/init.d/tomcat7 start
```

If you tail the log the following should print out:

```
martin@gravitonian:/var/lib/tomcat7$ tail -f logs/catalina.out
Apr 08, 2016 5:42:50 AM org.apache.catalina.core.StandardService startInternal
INFO: Starting service Catalina
Apr 08, 2016 5:42:50 AM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet Engine: Apache Tomcat/7.0.52 (Ubuntu)
Apr 08, 2016 5:42:50 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory /var/lib/tomcat7/webapps/ROOT
Apr 08, 2016 5:42:51 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Apr 08, 2016 5:42:51 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 938 ms
```

Access http://localhost:8080 and verify that it works. This will displays a minimal "It works" page by default.
Before moving on shutdown Tomcat:

```
$ sudo /etc/init.d/tomcat7 stop
```

Next step is to configure Tomcat for the Alfresco Solr 4 web application. I have unpacked the alfresco-solr4-5.1.f-config.zip file in the **~/Downloads/alfresco-solr4-5.1.f-config** directory.
Copy the unpacked files to a directory under **/var/lib/tomcat7** and set them up as accessible by the tomcat7 user:

```
/var/lib/tomcat7$ sudo mkdir data
/var/lib/tomcat7$ sudo chown tomcat7:tomcat7 data/
/var/lib/tomcat7$ cd data/
/var/lib/tomcat7/data$ sudo cp -r ~/Downloads/alfresco-solr4-5.1.f-config/* .
/var/lib/tomcat7/data$ sudo chown -R tomcat7:tomcat7 *
```

We should see something like this now:

```
martin@gravitonian:/var/lib/tomcat7/data$ ls -l
total 36
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 alfrescoModels
drwxr-xr-x 3 tomcat7 tomcat7 4096 Apr  8 05:56 archive-SpacesStore
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 conf
-rw-r--r-- 1 tomcat7 tomcat7  444 Apr  8 05:56 context.xml
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 lib
-rw-r--r-- 1 tomcat7 tomcat7  866 Apr  8 05:56 log4j-solr.properties
-rw-r--r-- 1 tomcat7 tomcat7  147 Apr  8 05:56 solr.xml
drwxr-xr-x 6 tomcat7 tomcat7 4096 Apr  8 05:56 templates
drwxr-xr-x 3 tomcat7 tomcat7 4096 Apr  8 05:56 workspace-SpacesStore
```

The Alfresco Solr configuration ZIP comes with the standard alfresco and archive core configurations that we need. We just need to configure each one as a Solr Slave and also tell it where it can find the Master Solr server.
Open up the **solrconfig.xml** file located in the **.../data/workspace-SpacesStore/conf** directory and change the `/replication` handler configuration as follows:

```
  <requestHandler name="/replication" class="solr.ReplicationHandler" >
      <lst name="slave">
        <str name="masterUrl">http://10.244.61.136:8080/solr4/alfresco</str>
        <str name="pollInterval">00:00:60</str>
      </lst>
  </requestHandler>
```

The masterUrl has the following format:

```
http://<master-solr-host>:<master-solr-port>/<solr webapp context>/<solr core name>
```

Now do the same change for the archived content configuration. Open up the **solrconfig.xml** file located in the **.../data/archive-SpacesStore/conf** and make the change as follows:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
    <lst name="slave">
      <str name="masterUrl">http://10.244.61.136:8080/solr4/archive</str>
      <str name="pollInterval">00:00:60</str>
    </lst>
</requestHandler>
```

Note that we are using a different core name in the URL.
Next things we need to configure in each core is where it should store its index data and where it can find the Alfresco Repository for Model tracking. Open up the **solrcore.properties** file located in the **.../data/workspace-SpacesStore/conf** directory and update the following properties (they do exist):

```
data.dir.root=/var/lib/tomcat7/data/index
alfresco.host=10.244.61.136
```

Make the same change in the **solrcore.properties** file located in the **.../data/archive-SpacesStore/conf** directory.

Now, set up Tomcat to deploy the Solr web application, the Alfresco Solr Configuration ZIP distribution comes with a web application context file that we can use:

```
/var/lib/tomcat7/data$ cd ..
/var/lib/tomcat7$ sudo cp data/context.xml conf/Catalina/localhost/solr4.xml
/var/lib/tomcat7$ cd conf/Catalina/localhost/
/var/lib/tomcat7/conf/Catalina/localhost$ sudo chown tomcat7:tomcat7 solr4.xml
```

Update the **solr4.xml** so paths match the installation, set the location of the Solr war file and the location of the Solr home directory:

```
<?xml version="1.0" encoding="utf-8"?>
<Context debug="0" crossContext="true">
  <Environment name="solr/home" type="java.lang.String" value="/var/lib/tomcat7/data" override="true"/>
  <Environment name="solr/model/dir" type="java.lang.String" value="/var/lib/tomcat7/data/model" override="true"/>
  <Environment name="solr/content/dir" type="java.lang.String" value="/var/lib/tomcat7/data/content" override="true"/>
</Context>
```

This should be all that is needed to setup Alfresco Solr 4 in a separate Tomcat.

The last thing to do before we can start Solr 4 is to copy over the customized Solr 4 WAR from the download dir (**note the name change of the WAR file so it matches the solr4.xml context file created earlier**):

```
/var/lib/tomcat7/conf/Catalina/localhost$ cd ../../../webapps
/var/lib/tomcat7/webapps$ sudo cp ~/Downloads/alfresco-solr4-5.1.f.war ./solr4.war
/var/lib/tomcat7/webapps$ sudo chown -R tomcat7:tomcat7 solr4.war
```

Then start Solr 4:

```
$ sudo /etc/init.d/tomcat7 start
```

If you see the following in the logs:

```
Exception in thread "SolrTrackingPool-alfresco-MetadataTracker-5" java.lang.OutOfMemoryError: Java heap space
```

Then you need to tweak the Java memory settings for Tomcat, the default 128MB is not going cut it. Set up the following JAVA_OPTS so you get enough memory to run Alfresco Solr:

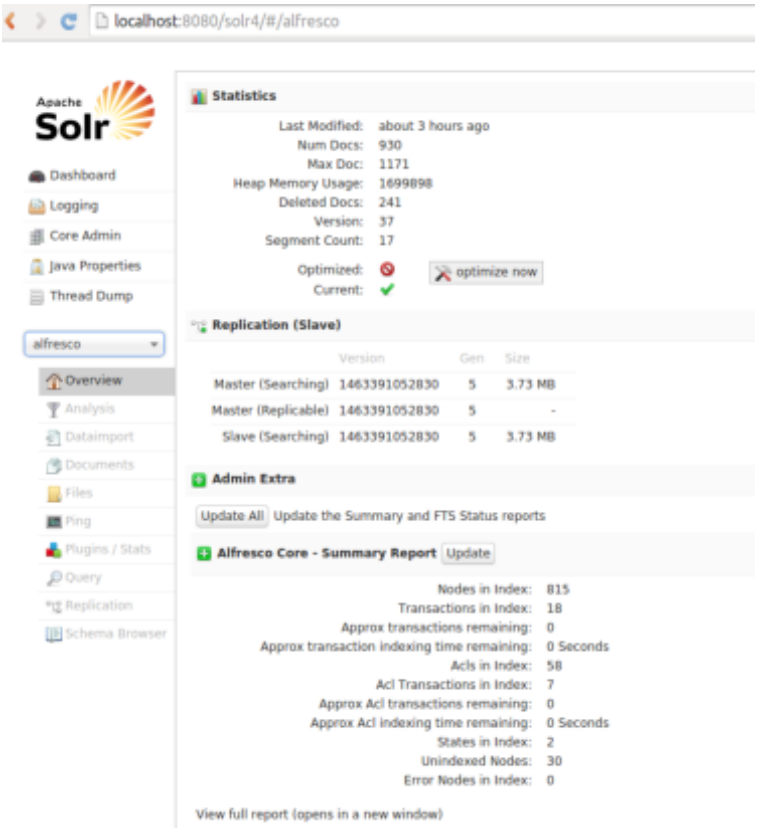
```
JAVA_OPTS="-Djava.awt.headless=true -Xmx512m -XX:+UseConcMarkSweepGC -XX:+UseParNewGC"
```

In a debian based system like Ubuntu I do this by updating the following file:

```
$ sudo gedit /etc/default/tomcat7
```

Note that it is not the actual boot script but the place where you can specify default values for variables.

Then restart Tomcat.
Starting up Solr and accessing the Admin UI at http://localhost:8080/solr4 should show something like this for the alfresco core:



Note that the Solr core is correctly identified as a Slave.
See the Alfresco online documentation for more information about how to set up replication.

Summary

This is all that is needed to set up Alfresco Solr replication. In a real scenario you would probably have more slaves to scale search and then also go via a load balancer when searching. If you want to move the Solr Master to a separate host then you would tell it about where the Alfresco Repository is running via the `alfresco.host` property in **solrcore.properties**.
The replication solution is good for scaling search but if you want to scale the size of the index then have a look at the sharding examples below.

Alfresco Solr Manual Sharding

In this section we will show an example of how to setup sharding with the Alfresco Community Edition (for more info see overview section).
The following nodes will participate in the setup:

- **Host 1**: with Alfresco Repository on Windows 7 (Tomcat 192.168.1.3:8080)
- **Host 2**: with Solr Shards 1 and 2 on Ubuntu (Tomcat 192.168.1.25:8080)
- **Host 3**: with Solr Shards 1b and 2b (replicas) on Ubuntu (Tomcat 192.168.1.188:8080)

Important: SSL is turned off between the repository server and the solr servers.

Setting up Alfresco Repository on Host 1 (192.168.1.3)

Download the latest Community Edition from https://www.alfresco.com/alfresco-community-download. Install Alfresco Community 5.1.f from the full installer (note. this installation includes Solr 4 but we are not going to use it).
Don't start Alfresco at this point.
Then disable the Solr webapp running locally by renaming **solr4.xml** in <alfrescoinstalldir>/tomcat/conf/Catalina/localhost to **solr4.xml.bak**. And remove the <alfrescoinstalldir>**/tomcat/webapps/solr4** directory and rename solr4.war to **solr4.war.bak**.
Now configure the Alfresco Server to use the stand-alone Solr servers with shards. The following properties in **alfresco-global.properties** tells Alfresco Repository where it can find the Solr Servers and how many shards to expect (including the number of replicas for each shard):

```
# Turn off https when sharding
# (i.e. the Tomcat installations that run the Solr shards
#  does not have https 8443 enabled)
solr.secureComms=none

### MANUAL SHARDING CONFIG ###
# Live content store (workspace://SpacesStore)
solr4.store.mappings.value.solrMappingAlfresco.nodeString=192.168.1.25:8080,192.168.1.188:8080
solr4.store.mappings.value.solrMappingAlfresco.numShards=2
solr4.store.mappings.value.solrMappingAlfresco.replicationFactor=2
# Archive content store (archive://SpacesStore)
solr4.store.mappings.value.solrMappingArchive.nodeString=192.168.1.25:8080,192.168.1.188:8080
solr4.store.mappings.value.solrMappingArchive.numShards=2
solr4.store.mappings.value.solrMappingArchive.replicationFactor=2
```

The `nodeString` property for each store defines the Solr server nodes that are available to the Alfresco Repository. The order in which these are specified is important. The first `<hostname>:<port>` expects to have shard 1, the second one shard 2 etc. The `numShards` property just tells Alfresco Repository how many shards the index is split up into.

And finally the `replicationFactor` property tells the Repository how many replicas there are of each shard core (i.e. shard instances).
The default properties for the Solr 4 sharding setup is defined as follows in **repository.properties**:

```
# Default SOLR 4 store mappings mappings
solr4.store.mappings=solrMappingAlfresco,solrMappingArchive
solr4.store.mappings.value.solrMappingAlfresco.httpClientFactory=solrHttpClientFactory
solr4.store.mappings.value.solrMappingAlfresco.baseUrl=/solr4/alfresco
solr4.store.mappings.value.solrMappingAlfresco.protocol=workspace
solr4.store.mappings.value.solrMappingAlfresco.identifier=SpacesStore
solr4.store.mappings.value.solrMappingArchive.httpClientFactory=solrHttpClientFactory
solr4.store.mappings.value.solrMappingArchive.baseUrl=/solr4/archive
solr4.store.mappings.value.solrMappingArchive.protocol=archive
solr4.store.mappings.value.solrMappingArchive.identifier=SpacesStore
```

Now start Alfresco Tomcat.
It should work to login to http://localhost:8080/share but searching will show connection failure in the log as no Solr server is up:

```
Caused by: java.net.ConnectException: Connection refused: connect
        at java.net.DualStackPlainSocketImpl.connect0(Native Method)
```

Setting up Solr Shards on Host 2 (192.168.1.25)

This involves installing Apache Tomcat, installing Solr, and creating shards.

Installing Tomcat and deploying Solr

The first thing we need to do is download and install Apache Tomcat 7 on the Linux box:

```
$ sudo apt-get install tomcat7
```

This will install Tomcat and its dependencies, such as Java, and it will also create the tomcat7 user. It also starts Tomcat with its default settings.
Now start Tomcat 7 as follows:

```
$ sudo /etc/init.d/tomcat7 start
```

If you tail the log the following should print out:

```
martin@gravitonian:/var/lib/tomcat7$ tail -f logs/catalina.out
Apr 08, 2016 5:42:50 AM org.apache.catalina.core.StandardService startInternal
INFO: Starting service Catalina
Apr 08, 2016 5:42:50 AM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet Engine: Apache Tomcat/7.0.52 (Ubuntu)
Apr 08, 2016 5:42:50 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory /var/lib/tomcat7/webapps/ROOT
Apr 08, 2016 5:42:51 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Apr 08, 2016 5:42:51 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 938 ms
```

Access http://localhost:8080 and verify that it works. This will displays a minimal "It works" page by default.
Before moving on shutdown Tomcat:

```
$ sudo /etc/init.d/tomcat7 stop
```

Next step is to configure Tomcat for the Apache Solr 4 web application. I have unpacked the alfresco-solr4-5.1.f-config.zip file in the **~/Downloads/alfresco-solr4-5.1.f-config** directory.
Copy the unpacked files to a directory under **/var/lib/tomcat7** and set them up as accessible by the tomcat7 user:

```
/var/lib/tomcat7$ sudo mkdir data
/var/lib/tomcat7$ sudo chown tomcat7:tomcat7 data/
/var/lib/tomcat7$ cd data/
/var/lib/tomcat7/data$ sudo cp -r ~/Downloads/alfresco-solr4-5.1.f-config/* .
/var/lib/tomcat7/data$ sudo chown -R tomcat7:tomcat7 *
```

We should see something like this now:

```
martin@gravitonian:/var/lib/tomcat7/data$ ls -l
total 36
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 alfrescoModels
drwxr-xr-x 3 tomcat7 tomcat7 4096 Apr  8 05:56 archive-SpacesStore
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 conf
-rw-r--r-- 1 tomcat7 tomcat7  444 Apr  8 05:56 context.xml
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 lib
-rw-r--r-- 1 tomcat7 tomcat7  866 Apr  8 05:56 log4j-solr.properties
-rw-r--r-- 1 tomcat7 tomcat7  147 Apr  8 05:56 solr.xml
drwxr-xr-x 6 tomcat7 tomcat7 4096 Apr  8 05:56 templates
drwxr-xr-x 3 tomcat7 tomcat7 4096 Apr  8 05:56 workspace-SpacesStore
```

The Alfresco Solr configuration ZIP comes with the standard alfresco and archive core configurations. We don't need them and we are also not going to be storing the fetched Alfresco models in the alfrescoModels directory. So delete them:

```
martin@gravitonian:/var/lib/tomcat7/data$ sudo rm -rf archive-SpacesStore/ workspace-SpacesStore/ alfrescoModels/
```

So we end up with:

```
martin@gravitonian:/var/lib/tomcat7/data$ ls -l
total 24
drwxr-xr-x 2 tomcat7 tomcat7 4096 May  5 11:50 conf
-rw-r--r-- 1 tomcat7 tomcat7  444 May  5 11:50 context.xml
drwxr-xr-x 2 tomcat7 tomcat7 4096 May  5 11:50 lib
-rw-r--r-- 1 tomcat7 tomcat7  866 May  5 11:50 log4j-solr.properties
-rw-r--r-- 1 tomcat7 tomcat7  147 May  5 11:50 solr.xml
drwxr-xr-x 6 tomcat7 tomcat7 4096 May  5 11:50 templates
```

Now, set up Tomcat to deploy the Solr web application, the Alfresco Solr Configuration ZIP distribution comes with a web application context file that we can use:

```
/var/lib/tomcat7/data$ cd ..
/var/lib/tomcat7$ sudo cp data/context.xml conf/Catalina/localhost/solr4.xml
```

```
/var/lib/tomcat7$ cd conf/Catalina/localhost/
/var/lib/tomcat7/conf/Catalina/localhost$ sudo chown tomcat7:tomcat7 solr4.xml
```

Update the **solr4.xml** so paths match the installation, set the location of the Solr war file and the location of the Solr home directory:

```
<?xml version="1.0" encoding="utf-8"?>
<Context debug="0" crossContext="true">
  <Environment name="solr/home" type="java.lang.String" value="/var/lib/tomcat7/data" override="true"/>
  <Environment name="solr/model/dir" type="java.lang.String" value="/var/lib/tomcat7/data/model" override="true"/>
  <Environment name="solr/content/dir" type="java.lang.String" value="/var/lib/tomcat7/data/content" override="true"/>
</Context>
```

This should be all that is needed to setup Alfresco Solr 4 in a separate Tomcat.

The last thing to do before we can start Solr 4 is to copy over the customized Solr 4 WAR from the download dir (**note the name change of the WAR file so it matches the solr4.xml context file created earlier**):

```
/var/lib/tomcat7/conf/Catalina/localhost$ cd ../../../webapps
/var/lib/tomcat7/webapps$ sudo cp ~/Downloads/alfresco-solr4-5.1.f.war ./solr4.war
/var/lib/tomcat7/webapps$ sudo chown -R tomcat7:tomcat7 solr4.war
```

Then start Solr 4:

```
$ sudo /etc/init.d/tomcat7 start
```

If you see the following in the logs:

```
Exception in thread "SolrTrackingPool-alfresco-MetadataTracker-5" java.lang.OutOfMemoryError: Java heap space
```

Then you need to tweak the Java memory settings for Tomcat, the default 128MB is not going cut it. Set up the following JAVA_OPTS so you get enough memory to run Alfresco Solr:

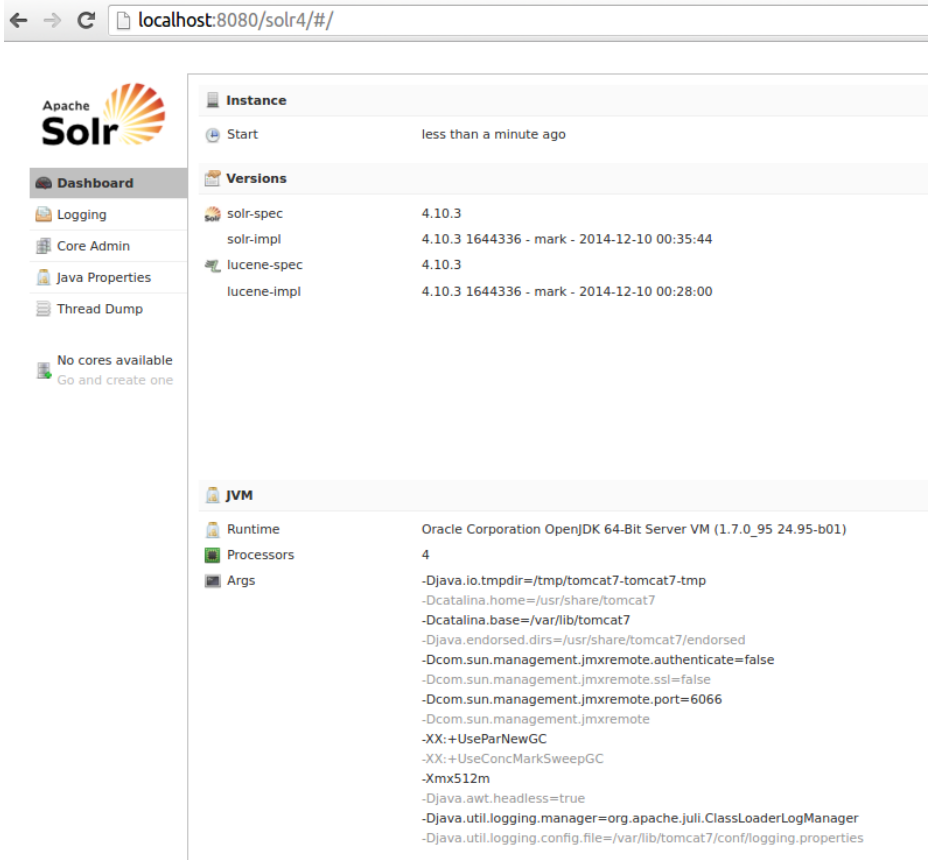
```
JAVA_OPTS="-Djava.awt.headless=true -Xmx512m -XX:+UseConcMarkSweepGC -XX:+UseParNewGC"
```

In a debian based system like Ubuntu I do this by updating the following file:

```
$ sudo gedit /etc/default/tomcat7
```

Note that it is not the actual boot script but the place where you can specify default values for variables.

Then restart Tomcat.
Starting up Solr and accessing the Admin UI at http://localhost:8080/solr4 should show something like this:



Note that there are no cores available yet.

Creating Solr Shards

Now, before we create the shards, make sure that the Alfresco Repository is running and that http://192.168.1.3:8080/alfresco is reachable for tracking communication from the Solr box. If it is not reachable, then you need to open up the firewall on the Windows box (or whatever box is running Alfresco Repository) for incoming HTTP connections on port 8080.
Create shards on the node. To do this we use the Alfresco Solr Core API as follows:

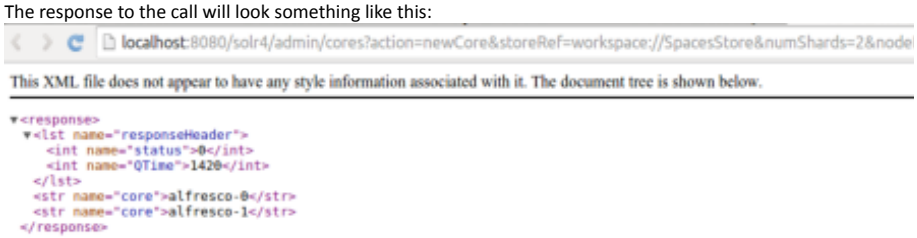
http://localhost:8080/solr4/admin/cores?action=newCore&storeRef=workspace://SpacesStore&numShards=2&nodeInstance=1&replicationFactor=2&numNodes=2&template=rerank&property.alfresco.host=192.168.1.3

The index (i.e. collection) consists of 2 shards and we want two copies each shard core so we specify `numShards` to `2` and `replicationFactor` to `2`. The configuration for these cores are copied from the **/var/lib/tomcat7/data/templates/rerank** directory as specified via the `template` parameter. The Alfresco Repository store that is tracked is specified with the `storeRef` parameter.

Note the last URL parameter, which defines a property that should be set in the **solrcore.properties** file. The value as specified in the URL (i.e. in this case we are configuring where the Solr core can connect to the Alfresco Repository to do tracking).

For more information about the URL parameters and what shard is created where see this section.

The response to the call will look something like this:

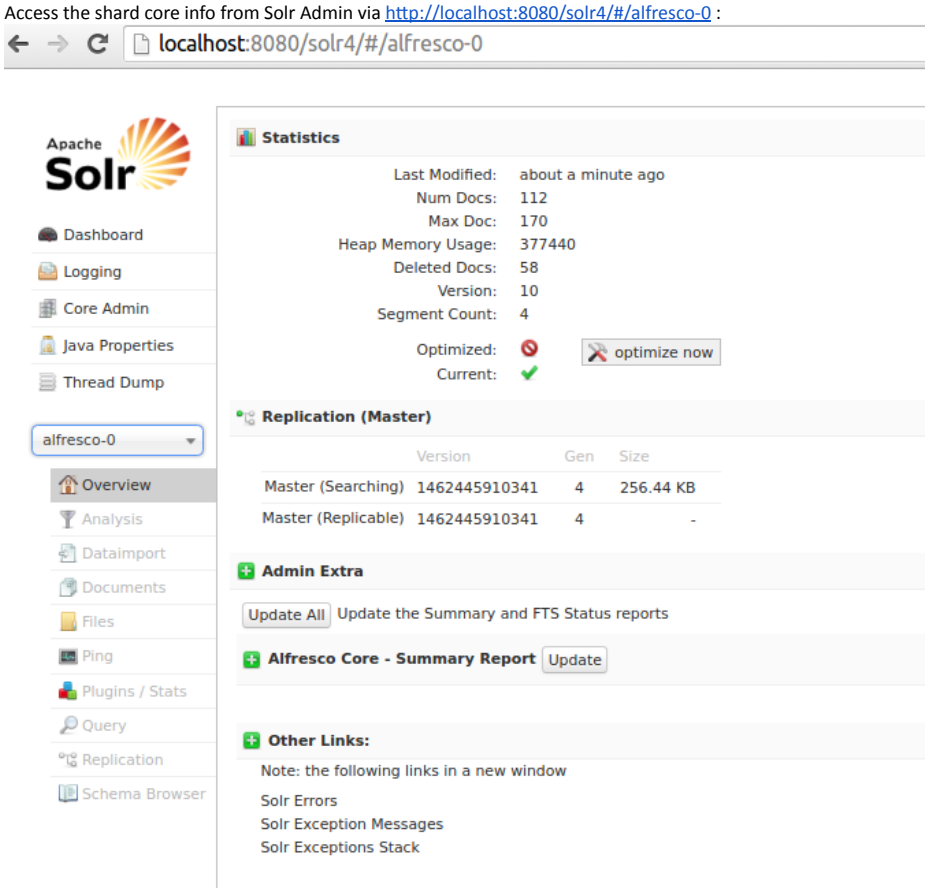


The response indicates that two shards (i.e. cores) with the name `alfresco-0` and `alfresco-1` have been created.

This generates the following folder structure under Solr Home (i.e. /var/lib/tomcat7/data):

```
rerank--workspace-SpacesStore--shards--2-x-2--node--1-of-2/
├── workspace-SpacesStore-0
|   ├── alfresco-0                      (SHARD 0 - instance 1)
|   |   └── index
|   |       ├── segments_1
|   |       ├── segments.gen
|   |       └── write.lock
|   ├── conf
|   |   ├── admin-extra.html
|   |   ├── admin-extra.menu-bottom.html
|   |   ├── admin-extra.menu-top.html
|   |   ├── elevate.xml
|   |   ├── lang
|   |   |   ├── contractions_ca.txt
|   |   |   ├── contractions_fr.txt
...
|   |   ├── protwords.txt
|   |   ├── _rest_managed.json
|   |   ├── schema.xml
|   |   ├── solrconfig.xml
|   |  ├── solrcore.properties
|   |   ├── spellings.txt
|   |   ├── ssl-keystore-passwords.properties
|   |   ├── ssl.repo.client.keystore
|   |   ├── ssl.repo.client.truststore
|   |   ├── ssl-truststore-passwords.properties
|   |   ├── stopwords.txt
|   |   └── synonyms.txt
|   └── core.properties
└── workspace-SpacesStore-1
    ├── alfresco-1                      (SHARD 1 - instance 1)
    |   └── index
    |       ├── segments_1
    |       ├── segments.gen
    |       └── write.lock
    ├── conf
    |   ├── admin-extra.html
    |   ├── admin-extra.menu-bottom.html
    |   ├── admin-extra.menu-top.html
    |   ├── elevate.xml
    |   ├── lang
    |   |   ├── contractions_ca.txt
    |   |   ├── contractions_fr.txt
...
    |   ├── protwords.txt
    |   ├── _rest_managed.json
    |   ├── schema.xml
    |   ├── solrconfig.xml
    |  ├── solrcore.properties
    |   ├── spellings.txt
    |   ├── ssl-keystore-passwords.properties
    |   ├── ssl.repo.client.keystore
    |   ├── ssl.repo.client.truststore
    |   ├── ssl-truststore-passwords.properties
    |   ├── stopwords.txt
    |   └── synonyms.txt
    └── core.properties
```

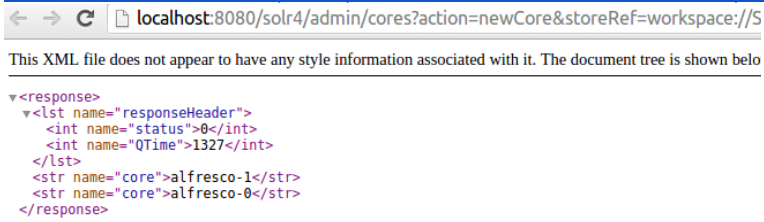
For detailed information about the files in this folder hierarchy see this article.
These two shard cores represent an instance of the collection we are creating that consists of two shards.
The configuration we get for these two indexes (cores) is copied from the **/var/lib/tomcat7/data/templates/rerank** directory, specified via the `template=rerank` URL parameter.
Access the shard core info from Solr Admin via http://localhost:8080/solr4/#/alfresco-0 :



Here we can see that the first shard called `alfresco-0` has started to track Alfresco repository and we currently got 112 docs in this shard. We can switch to the second shard by selecting `alfresco-1` in the left core selection box.
If you are not happy with the generated shards they can be removed and regenerated. Remove as follows:
http://localhost:8080/solr4/admin/cores?action=removeCore&coreName=alfresco-0
http://localhost:8080/solr4/admin/cores?action=removeCore&coreName=alfresco-1
Then generate the cores again via the `newCore` API call.

Setting up Solr Shards on Host 3 (192.168.1.188)

Do the same thing as for host 2 up until that the shards should be created with the `newCore` Alfresco Solr API call.
Then create the shards (replicas) with the following Alfresco Core API call (slightly different URL with `nodeInstance=2`):
http://localhost:8080/solr4/admin/cores?action=newCore&storeRef=workspace://SpacesStore&numShards=2&nodeInstance=2&replicationFactor=2&numNodes=2&template=rerank&property.alfresco.host=192.168.1.3



This generates the following folder structure under Solr Home (i.e. /var/lib/tomcat7/data):

```
rerank--workspace-SpacesStore--shards--2-x-2--node--2-of-2/
├── workspace-SpacesStore-0
|   ├── alfresco-0                    (SHARD 0 - instance 2)
|   |   └── index
|   |       ├── segments_1
|   |       ├── segments.gen
|   |       └── write.lock
|   ├── conf...
|   └── core.properties
└── workspace-SpacesStore-1
    ├── alfresco-1                    (SHARD 1 - instance 2)
    |   └── index
    |       ├── segments_1
    |       ├── segments.gen
```

```
|       └── write.lock
├── conf...
└── core.properties
```

So the second host will have replicas of the 2 shards we created on the other host.

Test that hosts are reachable

If things does not work as expected make sure that each host can reach the other host(s) that it needs to talk to.
Let's say you get the following error in the Alfresco Repository logs:

```
2016-05-05 10:40:05,701 ERROR [org.alfresco.repo.site.SiteServiceImpl] [http-apr-8080-exec-1] LuceneQueryParserException with findSites()
org.alfresco.repo.search.impl.lucene.LuceneQueryParserException: 04050041 Request failed 500 /solr4/alfresco-0/afts?
wt=json&fl=DBID%2Cscore&shards=10.244.61.62:8080%2Fsolr4%2Falfresco-0,10.244.51.19:8080%2Fsolr4%2Falfresco-
1&rows=5&df=TEXT&start=0&locale=en_GB&alternativeDic=DEFAULT_DICTIONARY&fq=%7B%21afts%7DAUTHORITY_FILTER_FROM_JSON&fq=%7B%21afts%7DTENANT_FILTER_FROM_JSON
```

Then this means that one or both of the Solr Shard servers is not reachable. From the Alfresco Repository host (Node 1) test that it can reach the 2 Solr nodes:

```
C:\alfresco>telnet 192.168.1.25 8080
C:\alfresco>telnet 192.168.1.188 8080
```

You should get a blank screen if the connection is going through.
Now from each one of the Solr hosts test that they can reach the Alfresco Repository host:

```
$ telnet 192.168.1.3 8080
```

Summary

So as we have seen, it is possible to try out the Alfresco Solr sharding features with the Community edition. Although the use-case is probably one that actually never practically exists, massive amounts of data with the load of a few users, maybe an archive solution without resilience and failover... Nevertheless, if you don't have access to the Alfresco Enterprise edition, then using the manual sharding configuration with the Community edition like this can be useful to get an understanding of how sharding works.
Because all is manually configured in a specific order it can be tricky when you get more than just a few shards. And if some node goes down or a Tomcat instance goes down, Alfresco Repository knows nothing about it. So in practice you need the Enterprise edition to get automatic shard registration and automatic shard state updates. Having an Alfresco Repository cluster also goes hand-in-hand with Solr Sharding, and you need the Enterprise edition to set up a cluster.

Alfresco Solr Dynamic Sharding

Having seen how to set up a manually configured sharding solution with Alfresco it is time to look at the Enterprise approach. Let's set up a cluster of Alfresco servers and have it talk to a sharded Solr index. As said before, clustering and dynamic sharding is only available in the Alfresco Enterprise edition (for more info about dynamic sharding see overview section).

The following nodes will participate in the setup:

- **Host 1 (Alfresco Cluster)**:
  - Alfresco Repository node 1 (Tomcat 192.168.1.25:8080)
  - Alfresco Repository node 2 (Tomcat 192.168.1.25:8081)
- **Host 2 (Sharded Index)**
  - Solr Shard 1a on Ubuntu (Tomcat 192.168.1.3:8080)
  - Solr Shard 2a on Ubuntu (Tomcat 192.168.1.3:8080)
- **Host 3 (Replica Sharded Index)**
  - Solr Shard 1b on Ubuntu (Tomcat 192.168.1.188:8080)
  - Solr Shard 2b on Ubuntu (Tomcat 192.168.1.188:8080)

So what we will set up here is a mini Alfresco Repository cluster on just one node, we are actually not digging into Alfresco clustering but instead the Solr sharding side of things, so making the Alfresco cluster install as small and painless as possible. Then we will have two hosts with a replicated 2 shard collection.
Important: SSL is turned off between the repository server and the solr servers.

Setting up Alfresco Cluster Node 1 on Host 1 (192.168.1.25)

Register and download the latest Alfresco Enterprise Edition from https://www.alfresco.com/products/one/trial. Install Alfresco Enterprise 5.1 from the full installer (note. this installation includes Solr 4 but we are not going to use it).
Don't start Alfresco at this point.
Then disable the Solr webapp running locally by renaming **solr4.xml** in <alfrescoinstalldir>/tomcat/conf/Catalina/localhost to **solr4.xml.bak**. And remove the <alfrescoinstalldir>**/tomcat/webapps/solr4** directory and rename solr4.war to **solr4.war.bak**.
Now configure the Alfresco Repository Server to expect a dynamic sharding solution where the different Solr servers report back shard information during tracking. The following properties should go into the **alfresco-global.properties** configuration file:

```
# Turn off https when sharding
# (i.e. the Tomcat installations that run the Solr shards
#  does not have https 8443 enabled)
solr.secureComms=none

# Use dynamic sharding
solr.useDynamicShardRegistration=true
```

There are also some other properties that might need tuning, but we are just going to keep the default values for them, which are as follows (so no need to include in alfresco-global.properties if you are not changing these values):

```
search.solrShardRegistry.purgeOnInit=true
search.solrShardRegistry.shardInstanceTimeoutInSeconds=300
search.solrShardRegistry.maxAllowedReplicaTxCountDifference=1000
```

The clustering functionality is **not** enabled in the Enterprise Trial license so you need to install a proper Enterprise license before proceeding (or you can continue without a cluster and just use one Enterprise Repository server):

```
/opt/alfresco5101$ cd tomcat/shared/classes/alfresco/extension/license/
/opt/alfresco5101/tomcat/shared/classes/alfresco/extension/license$ ls -l
total 8
-rw-rw-r-- 1 martin martin 992 May 13 14:49 Ent5.1-AllEnabled-Exp01012017.lic
```

Now start Alfresco:

```
/opt/alfresco5101$ ./alfresco.sh start
```

This creates the database and initializes the file system based content store.
It should now work to login to http://localhost:8080/share but searching does not work so you will see exceptions such as the following in the logs:

```
Failed to execute search:  +@cm\:modified:[2016\-5\-5T00\:00\:00.000 TO 2016\-5\-12T23\:59\:59.999] +@cm\:modifier:"admin" +TYPE:"cm:content" -
TYPE:"cm:systemfolder" -TYPE:"fm:forums" -TYPE:"fm:forum" -TYPE:"fm:topic" -TYPE:"fm:post" +(TYPE:"content" OR TYPE:"app:filelink" OR TYPE:"folder")
```

Stop Alfresco Tomcat before we move on to cluster node 2 so we don't get any changes to the database or content store while installing cluster node 2, which will share database and content store with node 1:

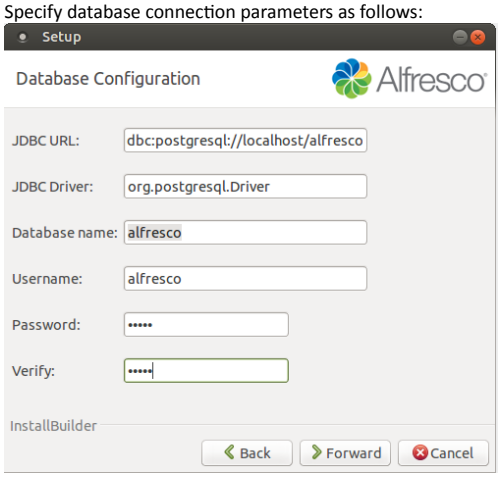
```
martin@gravitonian:/opt/alfresco5101$ ./alfresco.sh stop tomcat
/opt/alfresco5101/tomcat/scripts/ctl.sh : tomcat stopped
```

Note. the database should still be running, just Tomcat is stopped.

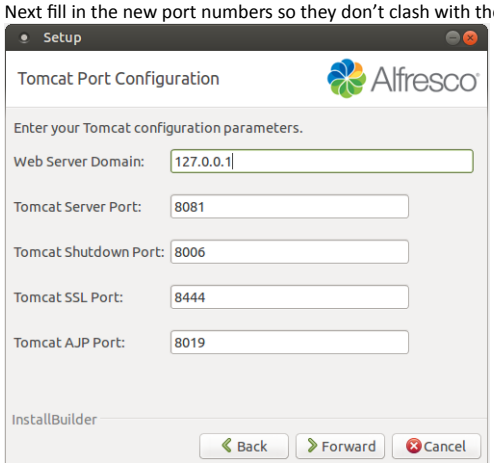
Setting up Alfresco Cluster Node 2 on Host 1 (192.168.1.25)

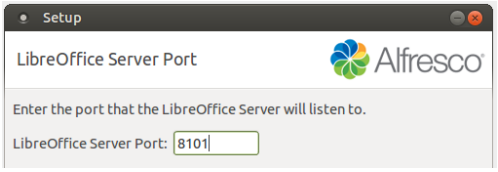
(*You can skip this section and jump to installing Solr node 1 if you don't have an Enterprise license*)
Install Alfresco Enterprise 5.1 **again** from the full installer. Choose **Advanced** Installation Type. During the install uncheck the database and Solr 4 as follows:

Specify database connection parameters as follows:

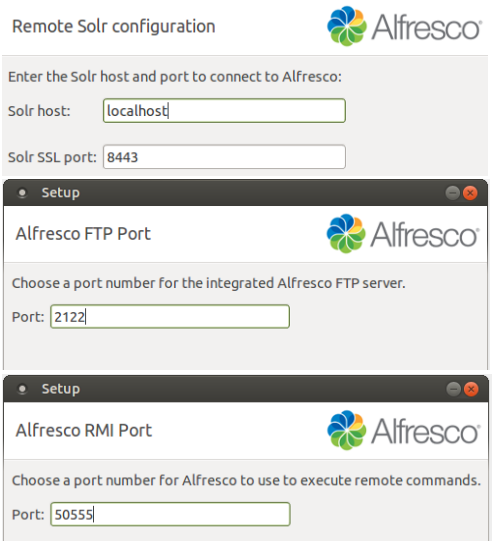


Use `admin` as password. This will connect to the same PostgreSQL database that was created by the first Alfresco Repository cluster node.
Next fill in the new port numbers so they don't clash with the first installation:





Keep the defaults for Solr host and port. The Alfresco repository will randomly select the Solr host to talk to during queries. The selected Solr host will then distribute the query to the other Solr hosts:



Don't start Alfresco at this point.
Then disable the Solr webapp running locally by renaming **solr4.xml** in <alfrescoinstalldir>/tomcat/conf/Catalina/localhost to **solr4.xml.bak**. And remove the <alfrescoinstalldir>**/tomcat/webapps/solr4** directory and rename solr4.war to **solr4.war.bak**.
Now configure this second Repository cluster node to:

- Use the same content store as the first Repository node (shared content store)
- Set a different Hazelcast port (distributed caching in cluster)
- Turn off SSL for Solr communication
- Use dynamic sharding

The following properties should go into the **alfresco-global.properties** configuration file or be updated:

```
dir.root=/opt/alfresco5101/alf_data
alfresco.hazelcast.port=5702
solr.secureComms=none
solr.useDynamicShardRegistration=true
```

The clustering functionality is **not** enabled in the Enterprise Trial license so you need to install a proper Enterprise license before proceeding (or you can continue without a cluster and just use one Enterprise Repository server):

```
/opt/alfresco5101-2$ cd tomcat/shared/classes/alfresco/extension/license/
/opt/alfresco5101-2/tomcat/shared/classes/alfresco/extension/license$ ls -l
total 8
-rw-rw-r-- 1 martin martin 992 May 13 14:49 Ent5.1-AllEnabled-Exp01012017.lic
```

Now start Alfresco (this will actually only start Apache Tomcat as we did not install a database for the second cluster node):

```
/opt/alfresco5101-2$ ./alfresco.sh start
```

This second node should use the same database and content store as the first node and start up quite quickly.
It should now work to login to http://localhost:8081/share but searching does not work so you will see exceptions such as the following in the logs:

```
Failed to execute search:  +@cm\:modified:[2016\-5\-5T00\:00\:00.000 TO 2016\-5\-12T23\:59\:59.999] +@cm\:modifier:"admin" +TYPE:"cm:content" -
TYPE:"cm:systemfolder" -TYPE:"fm:forums" -TYPE:"fm:forum" -TYPE:"fm:topic" -TYPE:"fm:post" +(TYPE:"content" OR TYPE:"app:filelink" OR TYPE:"folder")
```

Start Alfresco on Cluster Node 1 on Host 1 (192.168.1.25)

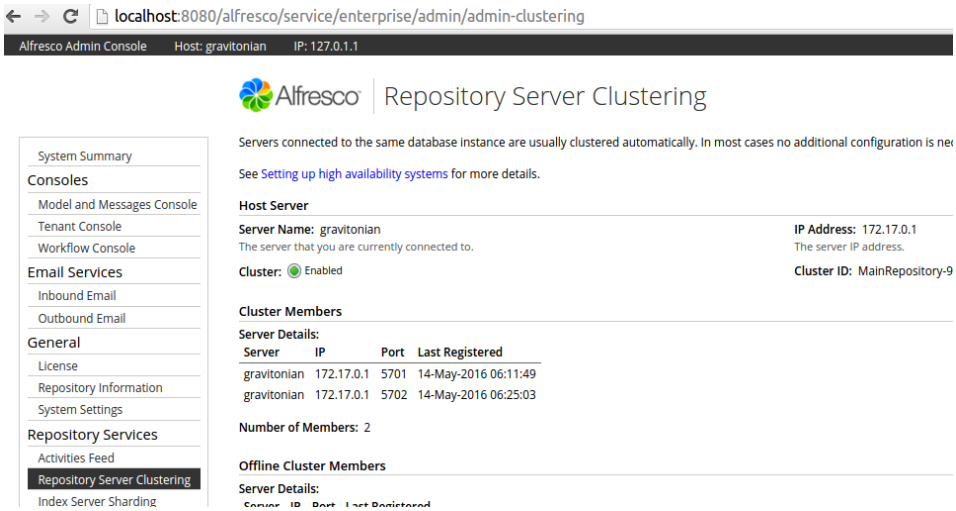
Now start Tomcat for the Alfresco Repository Node 1 again:

```
/opt/alfresco5101$ ./alfresco.sh start tomcat
```

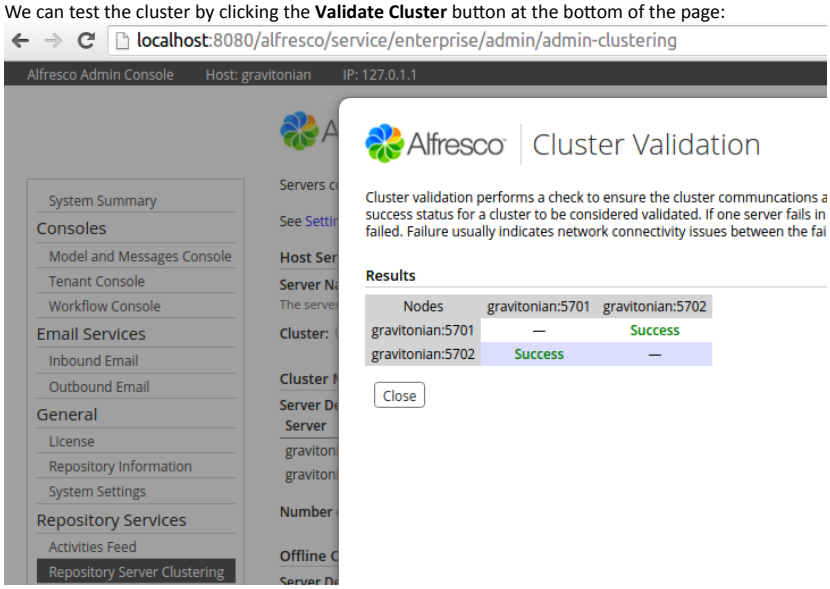
Test that it works to login to http://localhost:8080/share

Verify the cluster setup on Host 1 (192.168.1.25)

We should now have a mini Repository cluster running on Host 1. Verify that it is working properly via the Alfresco Enterprise Admin Console:



We can test the cluster by clicking the **Validate Cluster** button at the bottom of the page:



Setting up Solr shards on Host 2 (192.168.1.3)

This section goes through how to install Tomcat, Alfresco Solr, and shards on host 2.

Installing Apache Tomcat and Alfresco Solr 5.1 on Host 2 (192.168.1.3)

The first thing we need to do is download and install Apache Tomcat 7 on the Linux box:

```
$ sudo apt-get install tomcat7
```

This will install Tomcat and its dependencies, such as Java, and it will also create the tomcat7 user. It also starts Tomcat with its default settings.
Now start Tomcat 7 as follows (unless it is already started automatically):

```
$ sudo /etc/init.d/tomcat7 start
```

If you tail the log the following should print out:

```
martin@gravitonian:/var/lib/tomcat7$ tail -f logs/catalina.out
Apr 08, 2016 5:42:50 AM org.apache.catalina.core.StandardService startInternal
INFO: Starting service Catalina
Apr 08, 2016 5:42:50 AM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet Engine: Apache Tomcat/7.0.52 (Ubuntu)
Apr 08, 2016 5:42:50 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory /var/lib/tomcat7/webapps/ROOT
Apr 08, 2016 5:42:51 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Apr 08, 2016 5:42:51 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 938 ms
```

Access http://localhost:8080 and verify that it works. This will displays a minimal "It works" page by default.
Before moving on shutdown Tomcat:

```
$ sudo /etc/init.d/tomcat7 stop
```

Next step is to configure Tomcat for the Apache Solr 4 web application. I have unpacked the alfresco-solr4-5.1.1-config.zip (you need Enterprise access to Alfresco Nexus to download) file in the **~/Downloads/alfresco-solr4-5.1.1-config** directory.
Copy the unpacked files to a directory under **/var/lib/tomcat7** and set them up as accessible by the tomcat7 user:

```
/var/lib/tomcat7$ sudo mkdir data
/var/lib/tomcat7$ sudo chown tomcat7:tomcat7 data/
/var/lib/tomcat7$ cd data/
/var/lib/tomcat7/data$ sudo cp -r ~/Downloads/alfresco-solr4-5.1.1-config/* .
/var/lib/tomcat7/data$ sudo chown -R tomcat7:tomcat7 *
```

We should see something like this now:

```
martin@gravitonian:/var/lib/tomcat7/data$ ls -l
total 36
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 alfrescoModels
drwxr-xr-x 3 tomcat7 tomcat7 4096 Apr  8 05:56 archive-SpacesStore
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 conf
-rw-r--r-- 1 tomcat7 tomcat7  444 Apr  8 05:56 context.xml
drwxr-xr-x 2 tomcat7 tomcat7 4096 Apr  8 05:56 lib
-rw-r--r-- 1 tomcat7 tomcat7  866 Apr  8 05:56 log4j-solr.properties
-rw-r--r-- 1 tomcat7 tomcat7  147 Apr  8 05:56 solr.xml
drwxr-xr-x 6 tomcat7 tomcat7 4096 Apr  8 05:56 templates
drwxr-xr-x 3 tomcat7 tomcat7 4096 Apr  8 05:56 workspace-SpacesStore
```

The Alfresco Solr configuration ZIP comes with the standard alfresco and archive core configurations. We don't need them and we are also not going to be storing the fetched Alfresco models in the alfrescoModels directory. So delete them:

```
martin@gravitonian:/var/lib/tomcat7/data$ sudo rm -rf archive-SpacesStore/ workspace-SpacesStore/ alfrescoModels/
```

So we end up with:

```
martin@gravitonian:/var/lib/tomcat7/data$ ls -l
total 24
drwxr-xr-x 2 tomcat7 tomcat7 4096 May  5 11:50 conf
-rw-r--r-- 1 tomcat7 tomcat7  444 May  5 11:50 context.xml
drwxr-xr-x 2 tomcat7 tomcat7 4096 May  5 11:50 lib
-rw-r--r-- 1 tomcat7 tomcat7  866 May  5 11:50 log4j-solr.properties
-rw-r--r-- 1 tomcat7 tomcat7  147 May  5 11:50 solr.xml
drwxr-xr-x 6 tomcat7 tomcat7 4096 May  5 11:50 templates
```

Now, set up Tomcat to deploy the Solr web application, the Alfresco Solr Configuration ZIP distribution comes with a web application context file that we can use:

```
/var/lib/tomcat7/data$ cd ..
/var/lib/tomcat7$ sudo cp data/context.xml conf/Catalina/localhost/solr4.xml
/var/lib/tomcat7$ cd conf/Catalina/localhost/
/var/lib/tomcat7/conf/Catalina/localhost$ sudo chown tomcat7:tomcat7 solr4.xml
```

Update the **solr4.xml** so paths match the installation, set the location of the Solr war file and the location of the Solr home directory:

```
<?xml version="1.0" encoding="utf-8"?>
<Context debug="0" crossContext="true">
  <Environment name="solr/home" type="java.lang.String" value="/var/lib/tomcat7/data" override="true"/>
  <Environment name="solr/model/dir" type="java.lang.String" value="/var/lib/tomcat7/data/model" override="true"/>
  <Environment name="solr/content/dir" type="java.lang.String" value="/var/lib/tomcat7/data/content" override="true"/>
</Context>
```

This should be all that is needed to setup Alfresco Solr 4 in a separate Tomcat.

The last thing to do before we can start Solr 4 is to copy over the customized Solr 4 WAR from the download dir (**note the name change of the WAR file so it matches the solr4.xml context file created earlier**):

```
/var/lib/tomcat7/conf/Catalina/localhost$ cd ../../../webapps
/var/lib/tomcat7/webapps$ sudo cp ~/Downloads/alfresco-solr4-5.1.1.war ./solr4.war
/var/lib/tomcat7/webapps$ sudo chown -R tomcat7:tomcat7 solr4.war
```

Then start Solr 4:

```
$ sudo /etc/init.d/tomcat7 start
```

If you see the following in the logs:

```
Exception in thread "SolrTrackingPool-alfresco-MetadataTracker-5" java.lang.OutOfMemoryError: Java heap space
```

Then you need to tweak the Java memory settings for Tomcat, the default 128MB is not going cut it. Set up the following JAVA_OPTS so you get enough memory to run Alfresco Solr:

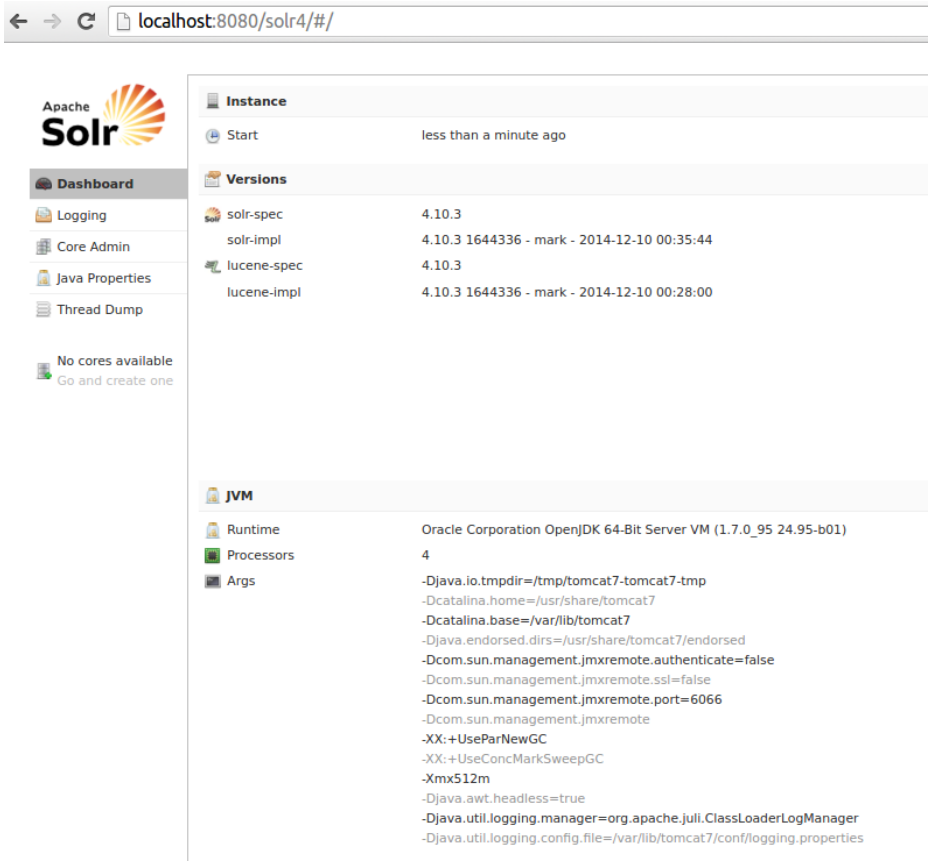
```
JAVA_OPTS="-Djava.awt.headless=true -Xmx512m -XX:+UseConcMarkSweepGC -XX:+UseParNewGC"
```

In a debian based system like Ubuntu I do this by updating the following file:

```
$ sudo gedit /etc/default/tomcat7
```

Note that it is not the actual boot script but the place where you can specify default values for variables.

Then restart Tomcat.
Starting up Solr and accessing the Admin UI at http://localhost:8080/solr4 should show something like this:

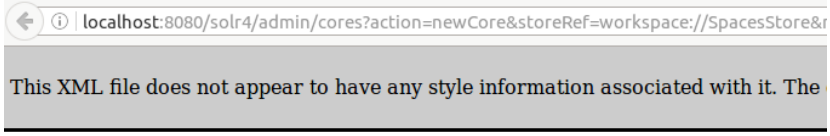


Note that there are no cores available yet.

Creating Solr shards on Host 2 (192.168.1.3)

Now, before we create shard 1 and 2, make sure that the Alfresco Repository is running and that http://192.168.1.25:8080/alfresco is reachable for tracking communication from the Solr box (we will use Repository cluster node 1 for tracking, in a real scenario you would go via a load balancer). If it is not reachable then check any firewalls between the boxes and make sure they let through HTTP connections on port 8080.
Create shards on the first Solr node (if we don't specify any `shardIds` in the URL Alfresco Solr will distribute them according to the formula that is explained in this section). To do this we use the Alfresco Solr Core API as follows:
http://localhost:8080/solr4/admin/cores?action=newCore&storeRef=workspace://SpacesStore&numShards=2&nodeInstance=1&replicationFactor=2&numNodes=2&template=rerank&property.alfresco.host=192.168.1.25

The index (i.e. collection) consists of 2 shards and we want two copies each shard core so we specify `numShards` and `replicationFactor` to `2`. The configuration for these cores are copied from the **/var/lib/tomcat7/data/templates/rerank** directory as specified via the template parameter. The Alfresco Repository store that is tracked is specified with the storeRef parameter.
Note the last URL parameter that is a property we want to go into the **solrcore.properties** file with the specied value (i.e. where is the Alfresco Repository that we want to track running).
For more information about the URL parameters see this section.
The response to the call will look something like this:



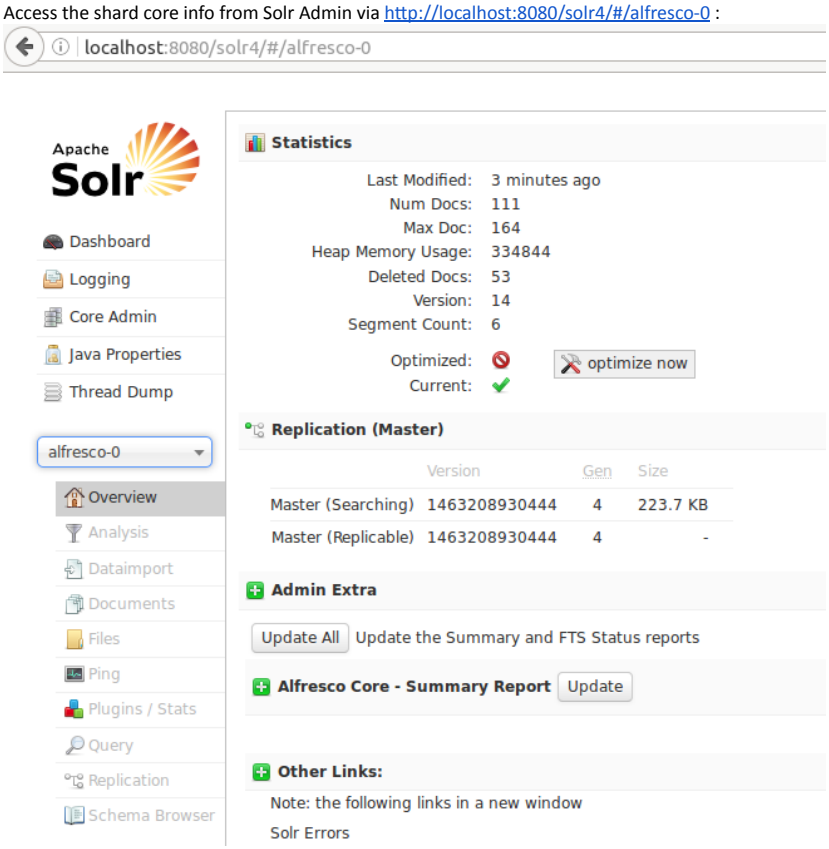

```
−<response>
  −<lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">3322</int>
    </lst>
    <str name="core">alfresco-0</str>
    <str name="core">alfresco-1</str>
  </response>
```

The response indicates that two shards (i.e. cores) with the name `alfresco-0` and `alfresco-1` have been created.
This generates the following folder structure under Solr Home (i.e. /var/lib/tomcat7/data):

```
rerank--workspace-SpacesStore--shards--2-x-2--node--1-of-2/
├── workspace-SpacesStore-0     (SHARD 1 - instance 1)
|   ├── alfresco-0
|   |   ├── conf
...
└── workspace-SpacesStore-1     (SHARD 2 - instance 1)
    ├── alfresco-1
```

```
    ├── conf
```

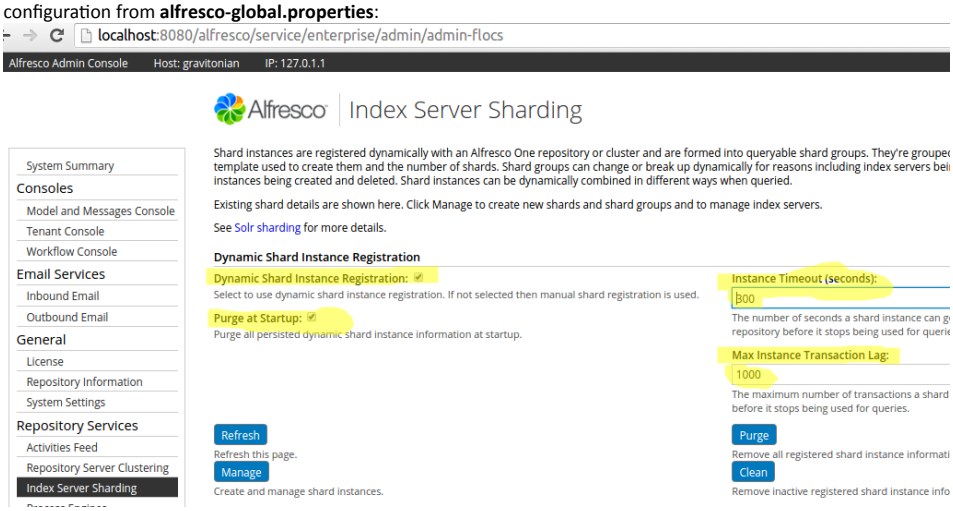
For detailed information about the files in this folder hierarchy see this article.
These two shard cores represent an instance of the collection we are creating that consists of two shards. They each track different ACLs.
The configuration we get for these two shards is copied from the **/var/lib/tomcat7/data/templates/rerank** directory, specified via the `template=rerank` URL parameter.
Access the shard core info from Solr Admin via http://localhost:8080/solr4/#/alfresco-0 :



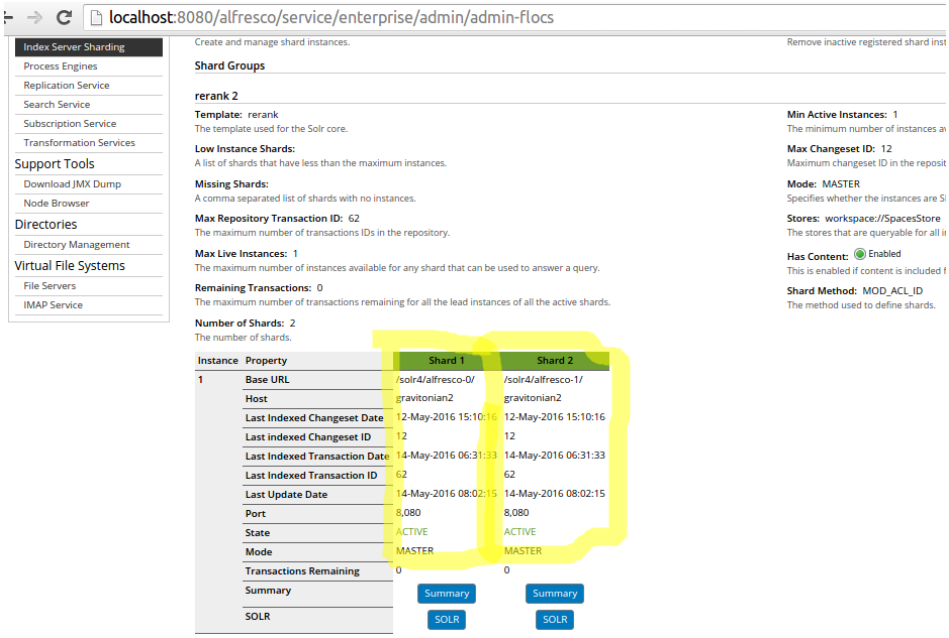
Here we can see that the alfresco-0 shard core has started to track Alfresco repository and we currently got 111 docs in **shard 1 - instance 1**. We can switch to **shard 2 - instance 1** by selecting alfresco-1 in the left core selection box.

Check the shard configuration via Repository Admin console (192.168.1.25)

Now check that the dynamic sharding works by checking current shard registrations via the Admin Console on the Repository cluster host. The first thing you see is just the configuration from **alfresco-global.properties**:



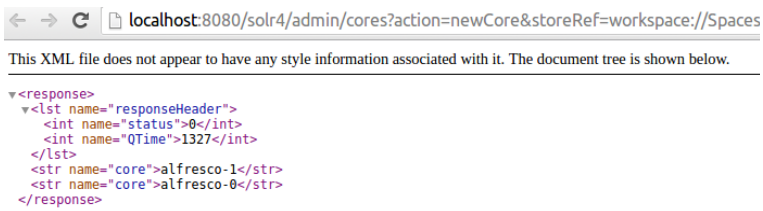
Scrolling down on the page reveals the shard registry information:



We can see here again that we got registrations for **shard 1 - instance 1** and **shard 2 - instance 1**.
In this example **gravitonian2** is the hostname with IP **192.168.1.3** (Solr node 1 on host 2).

Setting up Solr shards on Host 3 (192.168.1.188)

Install Apache Tomcat and Alfresco Solr 5.1 in the same way as for Solr node 1.
Then create shards with the following Alfresco Core API call (slightly different URL with `nodeInstance=2`):
http://localhost:8080/solr4/admin/cores?action=newCore&storeRef=workspace://SpacesStore&numShards=2&nodeInstance=2&replicationFactor=2&numNodes=2&template=rerank&property.alfresco.host=192.168.1.25

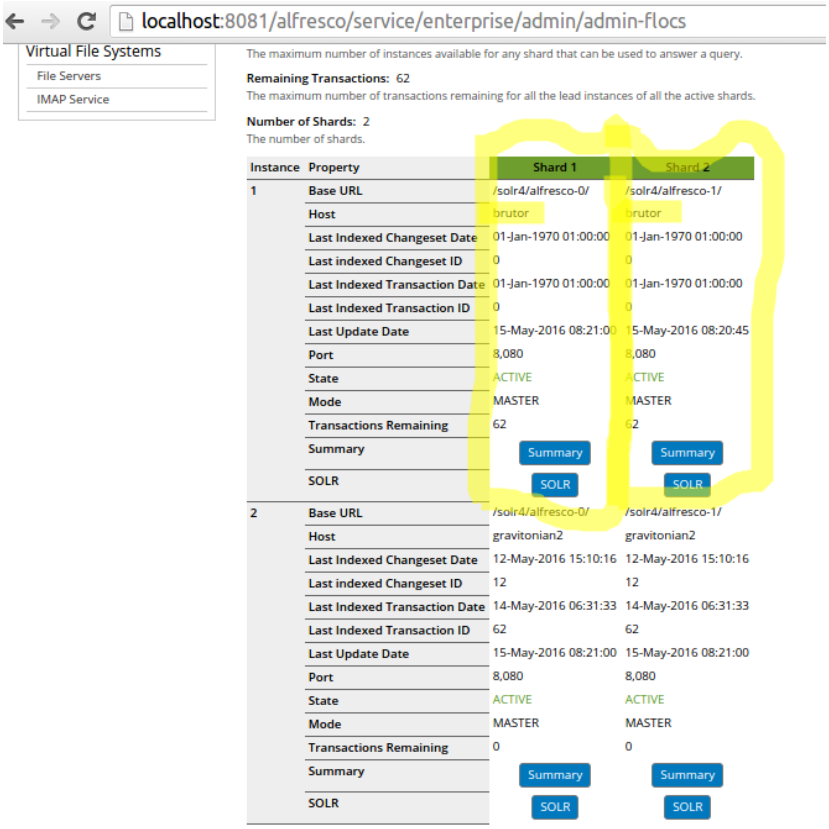


This generates the following folder structure under Solr Home (i.e. /var/lib/tomcat7/data):

```
rerank--workspace-SpacesStore--shards--2-x-2--node--2-of-2/
├── workspace-SpacesStore-0
|   ├── alfresco-0              (SHARD 1 - instance 2)
|   |   └── index
|   ├── conf...
|   └── core.properties
└── workspace-SpacesStore-1
    ├── alfresco-1              (SHARD 2 - instance 2)
    |   └── index
    ├── conf...
    └── core.properties
```

Check the shard configuration via Repository Admin console (192.168.1.25)

Now check that the dynamic sharding works by checking current shard registrations via the Admin Console on the Repository cluster host, the first thing you see is just the configuration from **alfresco-global.properties**:
Scrolling down on the page reveals the shard registry information:



We can see that we got 2 replicas for each shard spread over 2 hosts. The **brutor** shards are the ones we just created on host 3.
In this example **gravitonian2** is the hostname with IP **192.168.1.3** (Solr node 1 on host 2) and **brutor** is the hostname with IP **192.168.1.188** (Solr node 2 on host 3).

Summary

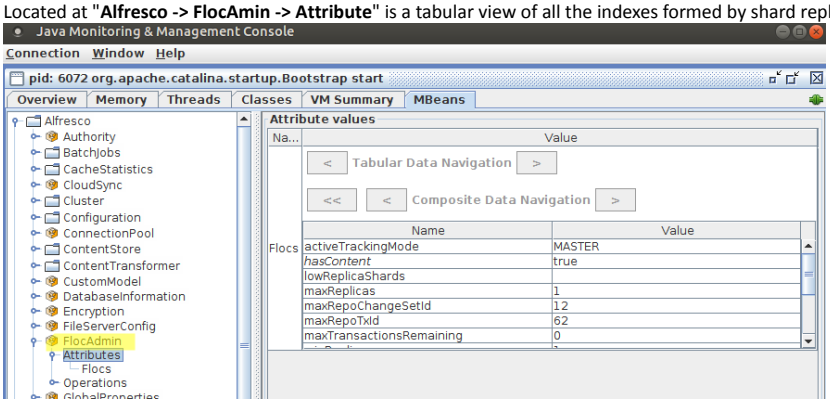
Setting up sharding with the Enterprise edition is quite easy as each node in the Repository cluster will be automatically updated with the shard instances that are available for querying. No manual shard configuration is needed in alfresco-global.properties.
The Enterprise Admin console is helpful in getting an understanding of the sharding and cluster setup.

One thing to note about this setup. Solr sharded requests include a post query phase to refine facet values. The queries generated by this phase can produce long URLs. It is recommended that you set the `maxHttpHeaderSize` in the Tomcat configuration. The exact value required depends on the facet field and how it is distributed.
The following value will be sufficient for 20 shards and 1e9 docs:

```
maxHttpHeaderSize="1048576"
```

Viewing Configuration via JMX

When running with the Enterprise edition there are a lot of information that can be extracted via JMX MBeans. This is typically very useful when you are setting up a system monitoring solution.
Located at "**Alfresco -> FlocAmin -> Attribute**" is a tabular view of all the indexes formed by shard replicas registering with any member of the Alfresco cluster:



You can move through the indexes using tabular navigation.

Floc/Index level information:

State

- activeTrackingMode - are replicas for the floc/index all SLAVE, MASTER or MIXED
- lowReplicaShards - a comma separated list of shards that have less that maxReplicas

- maxReplicas - the number of replicas for the shard with the maximum number of replicas
- maxRepoChangeSetId - the max changeset id in the repository
- maxRepoTxId - the max transaction id in the repository
- maxTransactions - the max transactions in any replica
- minReplicas - - the number of replicas for the shard with the miniumu number of replicas
- missingShards - a comma separated list of shards with no replicas

Detail

- shards - drill down for tabular data for each shard

Shard level information

You can navigate through each shard using tabular navigation

Configuration

- # - the shard number

State

- activeCount - the number of replicas currently able to answer queries
- activeTrackingMode - are replicas for the shard all SLAVE, MASTER or MIXED
- laggingCount - the number of replicas currently unable to answer queries as they are too far behind
- maxTransactionsRemaining - the maximum number of transactions left to index for any shard replica
- maxTxId - the max transaction id indexes by any replica
- silentCount - the number of replicas that are no longer tracking

Detail

- replicas - the detail for each instance in the shard

Replica Level Information

Location

- baseUrl - the URL to access the replica
- host - the host where the replica is located
- port - the port on the host where the replica is located

State

- lastIndexedChangeSetCommitTime
- lastIndexedChangeSetId
- lastIndexedTxCommitTime
- lastIndexedTxId
- lastUpdated - when the replica last reported state back to the repository
- state - the replica state ACTIVE, SILENT, LAGGING
- trackingMode - MASTER/SLAVE
- transactionsRemaining - the number of transactions remaining to be indexed.

Alfresco Solr related directory structure and files

Most of the files are described in the Searching with Alfresco 5.1 Community article.

<solr_home>/conf/shared.properties.sample

This file can be used to keep configuration that should be the same for all cores managed by a Solr Server installation. So any property in the **solrcore.properties** file that should have the same value for all cores can be set in this file ones it has been renamed to **shared.properties**.
More info about this file can be found in online docs.

How to upgrade from non sharded index to sharded index

To create a sharded solution from a non sharded installation follow these steps (same if you would like to change the number of shards):

1. Create machines to host Apache Solr servers with shards.
2. Install Alfresco Solr bundle as required.
3. Remove the existing Solr indexes from the install if they are not required.
4. Start Apache Solr on each host.
5. Create your new index shards and replicas as required. The shard cores will register and start tracking.
6. If there are two indexes for the same store the old one will be used until they are at the same state, then both will be used.
7. You can turn off any old indexes from tracking. Wait for the replicas to time out, for the new index to get ahead, or go to the JMX sharding operations and clear out all the registered shards and start again.
8. You have a new live sharded index

How to expand the search system to search more content sources

One thing that might come up in a project is the requirement to index and search an external content source (i.e. not files uploaded to Alfresco). This might be for example a website, a blog, wiki etc.
The question is, how would you do this? There are many approaches.

Importing content into existing store and core

One approach is to import the external content that should be indexed and searchable into the Alfresco Repository. Maybe into an “External Content” site. This content can then be searched in the usual way. And with a bit of customization to the search result screen we could even provide links to the external content, giving the end-user the opportunity to jump directly to it.
One way of getting the content into the site, if it is a webapp, is to use a Web crawler such as Apache Nutch. The crawler could use the REST API to upload text content for indexing.
Pros:

- No changes to the indexing and search mechanism is needed
- Minor updates to UI to link to external content

Cons:

- You have a site with content that is not really for end-user viewing
- Cannot “turn-off” searching in external content
- No way of dealing with the external content in a separate store so it can be indexed in a separate core and searched/scaled/sharded separately

Example implementation

TODO

Adding a store and a core

Another approach would be to create a separate repository store for the external content. And then also a separate Solr core.
So you end up with three cores:

- **Alfresco**: live content (workspace://SpacesStore)
- **Archived**: soft deleted content (archive://SpacesStore)
- **External**: external content (external://SpacesStore)

You would need a way of getting the content into the external repository store here too. So the Apache Nutch web crawler would be useful. Implementing the extra store and core would require quite a bit of coding compared to just adding content to an existing store. The solr search subsystem would need to be updated and the query logic would have to include the extra core.
You would get some benefits though in that you can scale and manage this content separately from the usual content in Alfresco.
Pros:

- Optionally include external content in searches
- Can scale search and indexing for external content separately from the usual content
- Clearer how the implementation is designed and works

Cons:

- Big implementation job

Example implementation

TODO