SolrCloud

{{{#!wiki red/solid ❌😞 **This page is outdated and you should read about SolrCloud at the Solr Reference Guide instead:** https://cwiki.apache.org/confluence/display/solr/SolrCloud. ❌😞
}}}

http://people.apache.org/~markrmiller/2shard4serverFull.jpg

SolrCloud

SolrCloud is the name of a set of new distributed capabilities in Solr. 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.

Look at the following 'Getting Started' section to quickly learn how to start up a cluster. There are 3 quick examples to follow, each showing how to startup a progressively more complicated cluster. After checking out the examples, consult the following sections for more detailed information as needed.

A little about SolrCores and Collections

On a single instance, Solr has something called a SolrCore that is essentially a single index. If you want multiple indexes, you create multiple SolrCores. With SolrCloud, a single index can span multiple Solr instances. This means that a single index can be made up of multiple SolrCore's on different machines. We call all of these SolrCores that make up one logical index a collection. A collection is a essentially a single index that spans many SolrCore's, both for index scaling as well as redundancy. If you wanted to move your 2 SolrCore Solr setup to SolrCloud, you would have 2 collections, each made up of multiple individual SolrCores.

Getting Started

Download Solr 4-Beta or greater: http://lucene.apache.org/solr/downloads.html

If you haven't yet, go through the simple Solr Tutorial to familiarize yourself with Solr. Note: reset all configuration and remove documents from the tutorial before going through the cloud features. Copying the example directories with pre-existing Solr indexes will cause document counts to be off.

Solr embeds and uses Zookeeper as a repository for cluster configuration and coordination - think of it as a distributed filesystem that contains information about all of the Solr servers

If you want to use a port other than 8983 for Solr, see the note about solr.xml under Parameter Reference below.

Example A: Simple two shard cluster

http://people.apache.org/~markrmiller/2shard2server.jpg

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.

```
rm -r example/solr/collection1/data/*

cp -r example example2
```

This command starts up a Solr server and bootstraps a new solr cluster.

```
cd example

java -Dbootstrap_confdir=./solr/collection1/conf -Dcollection.configName=myconf -DzkRun -DnumShards=2 -jar
start.jar
```

- `-DzkRun` causes an embedded zookeeper server to be run as part of this Solr server.
- `-Dbootstrap_confdir=./solr/collection1/conf` Since we don't yet have a config in zookeeper, this parameter causes the local configuration directory `./solr/conf` to be uploaded as the "myconf" config. The name "myconf" is taken from the "collection.configName" param below.
- `-Dcollection.configName=myconf` sets the config to use for the new collection. Omitting this param will cause the config name to default to "configuration1".
- `-DnumShards=2` the number of logical partitions we plan on splitting the index into.

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

You can see from the zookeeper browser that the Solr configuration files were uploaded under "myconf", and that a new document collection called "collection1" was created. Under collection1 is a list of shards, the pieces that make up the complete collection.

Now we want to start up our second server - it will automatically be assigned to shard2 because we don't explicitly set the shard id.

Then start the second server, pointing it at the cluster:

```
cd example2

java -Djetty.port=7574 -DzkHost=localhost:9983 -jar start.jar
```

- `-Djetty.port=7574` is just one way to tell the Jetty servlet container to use a different port.
- `-DzkHost=localhost:9983` points to the Zookeeper ensemble containing the cluster state. In this example we're running a single Zookeeper server embedded in the first Solr server. By default, an embedded Zookeeper server runs at the Solr port plus 1000, so 9983.

If you refresh the zookeeper browser, you should now see both shard1 and shard2 in collection1. View http://localhost:8983/solr/#/~cloud.

Next, index some documents. If you want to whip up some Java you can use the CloudSolrServer solrj impl and simply init it with the address to ZooKeeper . Or simply randomly choose which instance to add documents too - they will be automatically forwarded to where they belong:

```
cd exampledocs

java -Durl=http://localhost:7574/solr/collection1/update -jar post.jar ipod_video.xml

java -Durl=http://localhost:8983/solr/collection1/update -jar post.jar monitor.xml

java -Durl=http://localhost:7574/solr/collection1/update -jar post.jar mem.xml
```

And now, a request to either server results in a distributed search that covers the entire collection:

http://localhost:8983/solr/collection1/select?q=*:*

If at any point you wish to start over fresh or experiment with different configurations, you can delete all of the cloud state contained within zookeeper by simply deleting the solr/zoo_data directory after shutting down the servers.

Example B: Simple two shard cluster with shard replicas

`http://people.apache.org/~markrmiller/2shard4server.jpg`

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

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

```
cp -r example exampleB

cp -r example2 example2B
```

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

```
cd exampleB

java -Djetty.port=8900 -DzkHost=localhost:9983 -jar start.jar
```

```
cd example2B

java -Djetty.port=7500 -DzkHost=localhost:9983 -jar start.jar
```

Refresh the zookeeper browser page Solr Zookeeper Admin UI and verify that 4 solr nodes are up, and that each shard has two replicas.

Because we have been telling Solr that we want two logical shards, starting instances 3 and 4 are assigned to be additional replicas of those shards automatically.

Now send a query to any of the servers to query the cluster:

http://localhost:7500/solr/collection1/select?q=*:*

Send this query multiple times and observe the logs from the solr servers. You should be able to observe Solr load balancing the requests (done via LBHttpSolrServer ?) across replicas, using different servers to satisfy each request. There will be a log statement for the top-level request in the server the browser sends the request to, and then a log statement for each sub-request that are merged to produce the complete response.

To demonstrate fail-over for high availability, press CTRL-C in the window running any one of the Solr servers **except the instance running ZooKeeper**. (We'll talk about ZooKeeper redundancy in Example C.) Once that server instance terminates, send another query request to any of the remaining servers that are up. You should continue to see the full results.

SolrCloud can continue to serve results without interruption as long as at least one server hosts every shard. You can demonstrate this by judiciously shutting down various instances and looking for results. If you have killed all of the servers for a particular shard, requests to other servers will result in a 503 error. To return just the documents that are available in the shards that are still alive (and avoid the error), add the following query parameter: shards.tolerant=true

SolrCloud uses leaders and an overseer as an implementation detail. This means that some nodes/replicas will play special roles. You don't need to worry if the instance you kill is a leader or the cluster overseer - if you happen to kill one of these, automatic fail over will choose new leaders or a new overseer transparently to the user and they will seamlessly takeover their respective jobs. Any Solr instance can be promoted to one of these roles.

Example C: Two shard cluster with shard replicas and zookeeper ensemble

`http://people.apache.org/~markrmiller/2shard4server2.jpg`

The problem with example B is that while there are enough Solr servers to survive any one of them crashing, there is only one zookeeper server that contains the state of the cluster. If that zookeeper server crashes, distributed queries will still work since the solr servers remember the state of the cluster last reported by zookeeper. The problem is that no new servers or clients will be able to discover the cluster state, and no changes to the cluster state will be possible.

Running multiple zookeeper servers in concert (a zookeeper ensemble) allows for high availability of the zookeeper service. Every zookeeper server needs to know about every other zookeeper server in the ensemble, and a majority of servers are needed to provide service. For example, a zookeeper ensemble of 3 servers allows any one to fail with the remaining 2 constituting a majority to continue providing service. 5 zookeeper servers are needed to allow for the failure of up to 2 servers at a time.

For production, it's recommended that you run an external zookeeper ensemble rather than having Solr run embedded zookeeper servers. You can read more about setting up a zookeeper ensemble here. For this example, we'll use the embedded servers for simplicity.

First, stop all 4 servers and then clean up the zookeeper data directories for a fresh start.

```
rm -r example*/solr/zoo_data
```

We will be running the servers again at ports 8983,7574,8900,7500. The default is to run an embedded zookeeper server at hostPort+1000, so if we run an embedded zookeeper on the first three servers, the ensemble address will be `localhost:9983,localhost:8574,localhost:9900`.

As a convenience, we'll have the first server upload the solr config to the cluster. You will notice it block until you have actually started the second server. This is due to zookeeper needing a quorum before it can operate.

```
cd example

java -Dbootstrap_confdir=./solr/collection1/conf -Dcollection.configName=myconf -DzkRun -DzkHost=localhost:9983,
localhost:8574,localhost:9900 -DnumShards=2 -jar start.jar
```

```
cd example2

java -Djetty.port=7574 -DzkRun -DzkHost=localhost:9983,localhost:8574,localhost:9900 -jar start.jar
```

```
cd exampleB

java -Djetty.port=8900 -DzkRun -DzkHost=localhost:9983,localhost:8574,localhost:9900 -jar start.jar
```

```
cd example2B

java -Djetty.port=7500 -DzkHost=localhost:9983,localhost:8574,localhost:9900 -jar start.jar
```

Now since we are running three embedded zookeeper servers as an ensemble, everything can keep working even if a server is lost. To demonstrate this, kill the exampleB server by pressing CTRL+C in it's window and then browse to the Solr Zookeeper Admin UI to verify that the zookeeper service still works.

Note that when running on multiple hosts, you will need to set `-DzkRun=hostname:port` on each host to the exact name and port used in `-DzkHost` – the default `localhost` will not work.

ZooKeeper

Multiple Zookeeper servers running together for fault tolerance and high availability is called an ensemble. For production, it's recommended that you run an external zookeeper ensemble rather than having Solr run embedded servers. See the Apache ZooKeeper site for more information on downloading and running a zookeeper ensemble. More specifically, try Getting Started and ZooKeeper Admin. It's actually pretty simple to get going. You can stick to having Solr run ZooKeeper, but keep in mind that a ZooKeeper cluster is not easily changed dynamically. Until further support is added to ZooKeeper, changes are best done with rolling restarts. Handling this in a separate process from Solr will usually be preferable.

When Solr runs an embedded zookeeper server, it defaults to using the solr port plus 1000 for the zookeeper client port. In addition, it defaults to adding one to the client port for the zookeeper server port, and two for the zookeeper leader election port. So in the first example with Solr running at 8983, the embedded zookeeper server used port 9983 for the client port and 9984,9985 for the server ports.

In terms of trying to make sure ZooKeeper is setup to be very fast, keep a few things in mind: Solr does not use ZooKeeper intensively - optimizations may not be necessary in many cases. Also, while adding more ZooKeeper nodes will help some with read performance, it will slightly hurt write performance. Again, Solr does not really do much with ZooKeeper when your cluster is in a steady state. If you do need to optimize ZooKeeper, here are a few helpful notes:

1. ZooKeeper works best when it has a dedicated machine. ZooKeeper is a timely service and a dedicated machine helps ensure timely responses. A dedicated machine is not required however.
2. ZooKeeper works best when you put its transaction log and snap-shots on different disk drives.
3. If you do colocate ZooKeeper with Solr, using separate disk drives for Solr and ZooKeeper will help with performance.

Managing collections via the Collections API

The collections API let's you manage collections. Under the hood, it generally uses the CoreAdmin API to asynchronously (though Overseer) manage SolrCores on each server - it's essentially sugar for actions that you could handle yourself if you made individual CoreAdmin API calls to each server you wanted an action to take place on.

Create http://localhost:8983/solr/admin/collections?action=CREATE&name=mycollection&numShards=3&replicationFactor=4

About the params:

- **name**: The name of the collection to be created.
- **numShards**: The number of logical shards (sometimes called slices) to be created as part of the collection.
- **replicationFactor**: The number of copies of each document (or, the number of physical replicas to be created for each logical shard of the collection.) A replicationFactor of 3 means that there will be 3 replicas (one of which is normally designated to be the leader) for each logical shard. NOTE: in Solr 4.0, replicationFactor was the number of *additional* copies as opposed to the total number of copies.
- **maxShardsPerNode** : A create operation will spread numShards*replicationFactor shard-replica across your live Solr nodes - fairly distributed, and never two replica of the same shard on the same Solr node. If a Solr is not live at the point in time where the create operation is carried out, it will not get any parts of the new collection. To prevent too many replica being created on a single Solr node, use maxShardsPerNode to set a limit for how many replicas the create operation is allowed to create on each node - default is 1. If it cannot fit the entire collection numShards*replicationFactor replicas on you live Solrs it will not create anything at all.

- **createNodeSet**: If not provided the create operation will create shard-replica spread across all of your live Solr nodes. You can provide the "createNodeSet" parameter to change the set of nodes to spread the shard-replica across. The format of values for this param is "<node-name1>,<node-name2>,...,<node-nameN>" - e.g. "localhost:8983_solr,localhost:8984_solr,localhost:8985_solr"
- **collection.configName**: The name of the config (must be already stored in zookeeper) to use for this new collection. If not provided the create operation will default to the collection name as the config name.

⚠️ Solr4.2

About the params:

- **name**: The name of the collection alias to be created.
- **collections**: A comma-separated list of one or more collections to alias to.

Delete http://localhost:8983/solr/admin/collections?action=DELETE&name=mycollection

About the params:

- **name**: The name of the collection to be deleted.

Reload http://localhost:8983/solr/admin/collections?action=RELOAD&name=mycollection

About the params:

- **name**: The name of the collection to be reloaded.

Split Shard http://localhost:8983/solr/admin/collections?action=SPLITSHARD&collection=<collection_name>&shard=shardId

⚠️ Solr4.3

About the params:

- **collection**: The name of the collection
- **shard**: The shard to be split

This command cannot be used by clusters with custom hashing because such clusters do not rely on a hash range. It should only be used by clusters having "plain" or "compositeId" router.

The SPLITSHARD command will create two new shards by splitting the given shard's index into two pieces. The split is performed by dividing the shard's range into two equal partitions and dividing up the documents in the parent shard according to the new sub-ranges. This is a synchronous operation. The new shards will be named by appending _0 and _1 to the parent shard name e.g. if shard=shard1 is to be split, the new shards will be named as shard1_0 and shard1_1. Once the new shards are created, they are set active and the parent shard is set to inactive so that no new requests are routed to the parent shard.

This feature allows for seamless splitting and requires no down-time. The parent shard is not removed and therefore no data is removed. It is up to the user of the command to unload the shard using the new APIs in SOLR-4693 (under construction).

This feature was released with Solr 4.3 however due to bugs found after 4.3 release, it is recommended that you wait for release 4.3.1 before using this feature.

Collection Aliases

Aliasing allows you to create a single 'virtual' collection name that can point to one more real collections. You can update the alias on the fly.

CreateAlias http://localhost:8983/solr/admin/collections?action=CREATEALIAS&name=alias&collections=collection1,collection2,…

Creates or updates a given alias. Aliases that are used to send updates to should only map an alias to a single collection. Read aliases can map an alias to a single collection or multiple collections.

DeleteAlias http://localhost:8983/solr/admin/collections?action=DELETEALIAS&name=alias

Removes an existing alias.

Creating cores via CoreAdmin

New Solr cores may also be created and associated with a collection via CoreAdmin.

Additional cloud related parameters for the CREATE action:

- **collection** - the name of the collection this core belongs to. Default is the name of the core.
- **shard** - the shard id this core represents (Optional - normally you want to be auto assigned a shard id)
- **numShards** - the number of shards you want the collection to have - this is only respected on the first core created for the collection
- **collection.<param>=<value>** - causes a property of <param>=<value> to be set if a new collection is being created.
    - Use collection.configName=<configname> to point to the config for a new collection.

Example:

```
curl 'http://localhost:8983/solr/admin/cores?action=CREATE&name=mycore&collection=collection1&shard=shard2'
```

Distributed Requests

Query all shards of a collection (the collection is implicit in the URL):

```
http://localhost:8983/solr/collection1/select?
```

Query all shards of a compatible collection, explicitly specified:

```
http://localhost:8983/solr/collection1/select?collection=collection1_recent
```

Query all shards of multiple compatible collections, explicitly specified:

```
http://localhost:8983/solr/collection1/select?collection=collection1_NY,collection1_NJ,collection1_CT
```

Query specific shard ids of the (implicit) collection. In this example, the user has partitioned the index by date, creating a new shard every month:

```
http://localhost:8983/solr/collection1/select?shards=shard_200812,shard_200912,shard_201001
```

Explicitly specify the addresses of shards you want to query:

```
http://localhost:8983/solr/collection1/select?shards=localhost:8983/solr,localhost:7574/solr
```

Explicitly specify the addresses of shards you want to query, giving alternatives (delimited by `|`) used for load balancing and fail-over:

```
http://localhost:8983/solr/collection1/select?shards=localhost:8983/solr|localhost:8900/solr,localhost:7574
/solr|localhost:7500/solr
```

Required Config

All of the required config is already setup in the example configs shipped with Solr. The following is what you need to add if you are migrating old config files, or what you should not remove if you are starting with new config files.

schema.xml

You must have a *version* field defined:

```
<field name="_version_" type="long" indexed="true" stored="true" multiValued="false"/>
```

solrconfig.xml

You must have an UpdateLog defined - this should be defined in the updateHandler section.

```
    <!-- Enables a transaction log, currently used for real-time get.

         "dir" - the target directory for transaction logs, defaults to the

         solr data directory.  -->

    <updateLog>

      <str name="dir">${solr.data.dir:}</str>
      <!-- if you want to take control of the synchronization you may specify the syncLevel as one of the
           following where ''flush'' is the default. fsync will reduce throughput.
      <str name="syncLevel">flush|fsync|none</str>
      -->
```

```
    </updateLog>
```

You must have a replication handler called /replication defined:

```
    <requestHandler name="/replication" class="solr.ReplicationHandler" startup="lazy" />
```

You must have a realtime get handler called /get defined:

```
    <requestHandler name="/get" class="solr.RealTimeGetHandler">

      <lst name="defaults">

        <str name="omitHeader">true</str>

     </lst>

    </requestHandler>
```

You must have the admin handlers defined:

```
    <requestHandler name="/admin/" class="solr.admin.AdminHandlers" />
```

The DistributedUpdateProcessor is part of the default update chain and is automatically injected into any of your custom update chains. You can still explicitly add it yourself as follows:

```
   <updateRequestProcessorChain name="sample">

     <processor class="solr.LogUpdateProcessorFactory" />

     <processor class="solr.DistributedUpdateProcessorFactory"/>

     <processor class="my.package.UpdateFactory"/>

     <processor class="solr.RunUpdateProcessorFactory" />

   </updateRequestProcessorChain>
```

If you do not want the **DistributedUpdateProcessFactory** auto injected into your chain (say you want to use SolrCloud functionality, but you want to distribute updates yourself) then specify the following update processor factory in your chain: **NoOpDistributingUpdateProcessorFactory**

solr.xml

You must leave the admin path as the default:

```
    <cores adminPath="/admin/cores"
```

Re-sizing a Cluster

You can control cluster size by passing the numShards when you start up the first SolrCore in a collection. This parameter is used to auto assign which shard each instance should be part of. Any SolrCores that you start after starting numShards instances are evenly added to each shard as replicas (as long as they all belong to the same collection).

To add more SolrCores to your collection, simply keep starting new SolrCores up. You can do this at any time and the new SolrCore will sync up its data with the current replicas in the shard before becoming active.

If you want to start your cluster on fewer machines and then expand over time beyond just adding replicas, you can choose to start by hosting multiple shards per machine (using multiple SolrCores) and then later migrate shards onto new machines by starting up a new replica for a given shard and eventually removing the shard from the original machine.

⚠️ Solr4.3 The new "SPLITSHARD" collection API can be used to split an existing shard into two shards containing exactly half the range of the parent shard each. More details can be found under the "Managing collections via the Collections API" section.

Near Realtime Search

If you want to use the Near Realtime search support, you will probably want to enable auto soft commits in your solrconfig.xml file before putting it into zookeeper. Otherwise you can send explicit soft commits to the cluster as you desire. See NearRealtimeSearch

Parameter Reference

Cluster Params

numShards	Defaults to 1	The number of shards to hash documents to. There will be one leader per shard and each leader can have N replicas.

SolrCloud Instance Params

These are set in solr.xml, but by default they are setup in solr.xml to also work with system properties. Important note: the hostPort value found here will be used (via zookeeper) to inform the rest of the cluster what port each Solr instance is using. The default port is 8983. The example solr.xml uses the jetty.port system property, so if you want to use a port other than 8983, either you have to set this property when starting Solr, or you have to change solr.xml to fit your particular installation. If you do not do this, the cluster will think all your Solr servers are using port 8983, which may not be what you want.

host	Defaults to the first local host address found	If the wrong host address is found automatically, you can over ride the host address with this param.
hostPort	Defaults to the jetty.port system property	The port that Solr is running on - by default this is found by looking at the jetty.port system property.
hostContext	Defaults to solr	The context path for the Solr webapp. (Note: in Solr 4.0, it was mandatory that the hostContext not contain "/" or "_" characters. Begining with Solr 4.1, this limitation was removed, and it is recomended that you specify the begining slash. When running in the example jetty configs, the "hostContext" system property can be used to control both the servlet context used by jetty, and the hostContext used by SolrCloud – eg: `-DhostContext=/solr`)

SolrCloud Instance ZooKeeper Params

zkRun	Defaults to localhost:<solrPort+1001>	Causes Solr to run an embedded version of ZooKeeper. Set to the address of ZooKeeper on this node - this allows us to know who 'we are' in the list of addresses in the `zkHost` connect string. Simply using `-DzkRun` gets you the default value. Note this must be one of the exact strings from `zkHost`; in particular, the default `localhost` will not work for a multi-machine ensemble.
zkHost	No default	The host address for ZooKeeper - usually this should be a comma separated list of addresses to each node in your ZooKeeper ensemble.
zkClientTimeout	Defaults to 15000	The time a client is allowed to not talk to ZooKeeper before having it's session expired.

zkRun and zkHost are setup using system properties. zkClientTimeout is setup in solr.xml, but default, can also be set using a system property.

SolrCloud Core Params

shard	The shard id. Defaults to being automatically assigned based on numShards	Allows you to specify the id used to group SolrCores into shards.

shard can be configured in solr.xml for each core element as an attribute.

Getting your Configuration Files into ZooKeeper

Config Startup Bootstrap Params

There are two different ways you can use system properties to upload your initial configuration files to ZooKeeper the first time you start Solr. Remember that these are meant to be used only on first startup or when overwriting configuration files - everytime you start Solr with these system properties, any current configuration files in ZooKeeper may be overwritten when 'conf set' names match.

1. Look at solr.xml and upload the conf for each SolrCore found. The 'config set' name will be the collection name for that SolrCore, and collections will use the 'config set' that has a matching name.

bootstrap_conf	No default	If you pass -Dbootstrap_conf=true on startup, each SolrCore you have configured will have it's configuration files automatically uploaded and linked to the collection that SolrCore is part of

2. Upload the given directory as a 'conf set' with the given name. No linking of collection to 'config set' is done. However, if only one 'conf set' exists, a collection will auto link to it.

bootstrap_confdir	No default	If you pass -bootstrap_confdir=<directory> on startup, that specific directory of configuration files will be uploaded to ZooKeeper with a 'conf set' name defined by the below system property, collection.configName
collection.configName	Defaults to configuration1	Determines the name of the conf set pointed to by bootstrap_confdir

Command Line Util

The ZkCLI tool (aka `zkcli.sh` and `zkcli.bar`) also lets you upload config to ZooKeeper. It allows you to do it the same two ways that you can above. It also provides a few other commands that let you link collection sets to collections, make ZooKeeper paths or clear them, as well as download configs from ZooKeeper to the local filesystem.

Details on using the ZkCLI command line tool and the options it supports can be found in the Solr Ref Guide.

Examples

See: https://cwiki.apache.org/confluence/display/solr/Command+Line+Utilities#CommandLineUtilities-ZooKeeperCLIExamples

Scripts

See: https://cwiki.apache.org/confluence/display/solr/Command+Line+Utilities#CommandLineUtilities-Scripts

Zookeeper chroot

If you are already using Zookeeper for other applications and you want to keep the ZNodes organized by application, or if you want to have multiple separated SolrCloud clusters sharing one Zookeeper ensemble you can use Zookeeper's "chroot" option. From Zookeeper's documentation: http://zookeeper.apache.org/doc/r3.3.6/zookeeperProgrammers.html#ch_zkSessions

```
An optional "chroot" suffix may also be appended to the connection string. This will run the client commands
while interpreting all paths relative to this root (similar to the unix chroot command). If used the example
would look like: "127.0.0.1:4545/app/a" or "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002/app/a" where the
client would be rooted at "/app/a" and all paths would be relative to this root - ie getting/setting/etc... "
/foo/bar" would result in operations being run on "/app/a/foo/bar" (from the server perspective).
```

To use this Zookeeper feature, simply start Solr with the "chroot" suffix in the zkHost parameter. For example:

```
java -DzkHost=localhost:9983/foo/bar -jar start.jar
```

or

```
java -DzkHost=zoo1:9983,zoo2:9983,zoo3:9983/foo/bar -jar start.jar
```

NOTE: With Solr 4.0 you'll need to create the initial path in Zoookeeper before starting Solr. Since Solr 4.1, the initial path will automatically be created if you are using either *bootstrap_conf* or *boostrap_confdir*.

Known Limitations

A small number of Solr search components do not support DistributedSearch. In some cases, a component may never get distributed support, in other cases it may just be a matter of time and effort. All of the search components that do not yet support standard distributed search have the same limitation with SolrCloud. You can pass distrib=false to use these components on a single SolrCore.

The Grouping feature only works if groups are in the same shard. You must use the custom sharding feature to use the Grouping feature.

If upgrading an existing Solr instance instance running with SolrCloud from Solr 4.0 to 4.1, be aware that the way the name_node parameter is defined has changed. This may cause a situation where the name_node uses the IP address of the machine instead of the server name, and thus SolrCloud is not aware of the existing node. If this happens, you can manually edit the host parameter in solr.xml to refer to the server name, or set the host in your system environment variables (since by default solr.xml is configured to inherit the host name from the environment variables). See also the section Core Admin and Configuring solr.xml for more information about the host parameter.

Glossary

	A single search index.

Collection:	
Shard:	A logical section of a single collection (also called Slice). Sometimes people will talk about "Shard" in a physical sense (a manifestation of a logical shard)
Replica:	A physical manifestation of a logical Shard, implemented as a single Lucene index on a SolrCore
Leader:	One Replica of every Shard will be designated as a Leader to coordinate indexing for that Shard
SolrCore:	Encapsulates a single physical index. One or more make up logical shards (or slices) which make up a collection.
Node:	A single instance of Solr. A single Solr instance can have multiple SolrCores that can be part of any number of collections.
Cluster:	All of the nodes you are using to host SolrCores.

FAQ

- **Q:** I'm seeing lot's of session timeout exceptions - what to do?
  - **A:** Try raising the ZooKeeper session timeout by editing solr.xml - see the zkClientTimeout attribute. The minimum session timeout is 2 times your ZooKeeper defined tickTime. The maximum is 20 times the tickTime. The default tickTime is 2 seconds. You should avoiding raising this for no good reason, but it should be high enough that you don't see a lot of false session timeouts due to load, network lag, or garbage collection pauses. The default timeout is 15 seconds, but some environments might need to go as high as 30-60 seconds.
- **Q:** How do I use SolrCloud, but distribute updates myself?
  - **A:** Add the following UpdateProcessorFactory somewhere in your update chain: **NoOpDistributingUpdateProcessorFactory**
- **Q:** What is the difference between a Collection and a SolrCore?
  - **A:** In classic single node Solr, a SolrCore is basically equivalent to a Collection. It presents one logical index. In SolrCloud, the SolrCore's on multiple nodes form a Collection. This is still just one logical index, but multiple SolrCores host different 'shards' of the full collection. So a SolrCore encapsulates a single physical index on an instance. A Collection is a combination of all of the SolrCores that together provide a logical index that is distributed across many nodes.
- **Q:** Does SolrCloud provide the ability to dynamically add shards to a Collection ?
  - **A:** SolrCloud supports the ability to add New Shards (or DELETE existing shards) to your index (whenever you want) via the "implicit router" configuration in the CREATECOLLECTION API.

Lets say - you have to index all "Audit Trail" data of your application into Solr. New Data gets added every day. You might most probably want to shard by year.

You could do something like the below during the initial setup of your collection:

admin/collections?
action=CREATE&
name=AuditTrailIndex&
router.name=implicit&
shards=2010,2011,2012,2013,2014&
router.field=year

The above command:
a) Creates 5 shards - one each for the current and the last 4 years 2010,2011,2012,2013,2014

b) Routes data to the correct shard based on the value of the "year" field (specified as router.field)

In December 2014, you might add a new shard in preparation for 2015 using the CREATESHARD API (part of the Collections API) - Do something like:

/admin/collections?
action=CREATESHARD&
shard=2015&
collection=AuditTrailIndex

The above command creates a new shard on the same collection.

When its 2015, all data will get automatically indexed into the "2015" shard assuming your data has the "year" field populated correctly to 2015.

In 2015, if you think you don't need the 2010 shard (based on your data retention requirements) - you could always use the DELETESHARD API to do so:

/admin/collections?
action=DELETESHARD&
shard=2015&
collection=AuditTrailIndex

This solution only works if you used the "implicit router" when creating your collection. Does NOT work when you use the default "compositeId router" - i.e. collections created with the numshards parameter.