System Administrator

LIFERAY Training

6.1 Clustering Liferay

6.1.1 Why a Cluster?

- For larger installations, you need a clustered configuration in order to handle a large amount of traffic on your Site.
- A cluster allows you to distribute your Site's traffic over several machines.
- This allows your Site to handle more web traffic at a faster pace than would be possible with a single machine.
- A clustered configuration also ensures the high availability of your Liferay server: when one of your servers goes down, the other servers continue to serve your Site.
- Liferay is designed for high availability situations and scales well when clustered.
- **Clustering is only available in Liferay DXP.**

6.1.2 Overview

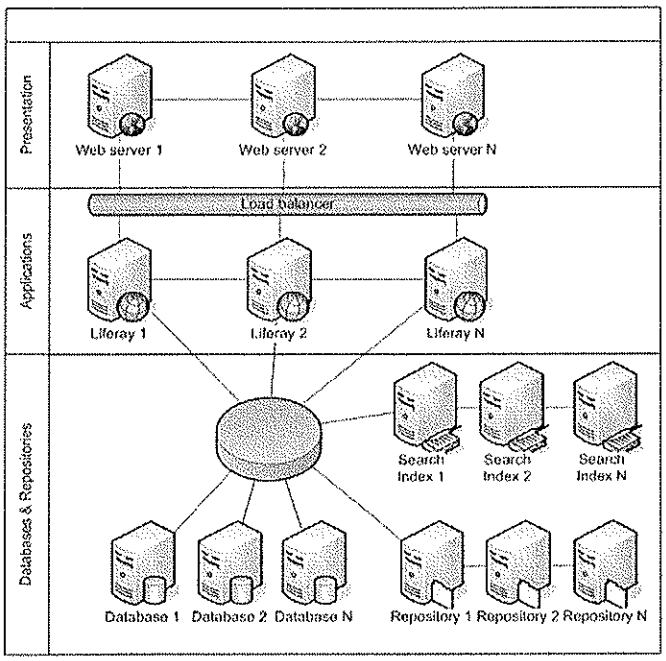
- Logical view of typical clustered architecture

(Figure 6.1, page 185)

6.1.3 Principles of Liferay Clustering

- All nodes should be pointing to the same Liferay database or database cluster.
- Configure the file repository for clustering.
- Configure indexing (Elasticsearch, Solr) of clustering.
- Load balancing: often using hardware can also be done with an HTTPD server
- Plugins must be deployed to all of the nodes.

Figure 6.1:



  - Some application servers support farms, where deploying an application to one node transmits it to all of the other nodes.

- Optional: Configure the application server to replicate sessions.

6.1.4 Configuring the database

- In a Liferay Cluster, all nodes must access the same database.
- This will need to be done in the `portal-setup-wizard.properties` or `portal-ext.properties` files of each node.
- For Liferay, it doesn't matter if the database is clustered, since it's accessed as a single entity.

6.1.5 Enabling Liferay clustering

- Multiple facets of clustering Liferay are handled by a tool called **Cluster Link**.

- The Cluster Link provides optimized communication across the cluster.

- It's simple enough to enable: just add the following line to all nodes' `portal-ext.properties` files:

```
cluster.link.enabled=true
```

- Cluster Link establishes a UDP-based multicast communication between the target nodes by default. TCP-based, unicast configuration is also available by setting another `cluster.link`-related attribute in `portal-ext.properties` (set `tcp.xml` accordingly).

```
cluster.link.channel.properties.control=myehcache/tcp.xml
cluster.link.channel.properties.transport.0=myehcache/tcp.xml
```

- Requirements: `com.liferay.portal.cluster.multiple.jar`, an OSGi bundle which provides the implementation of core clustering features, must be installed and started.

6.1.6 Quartz cluster mode

- Enabling Cluster Link automatically enables the Quartz scheduler clustered mode.

- Jobs running by scheduler will have two choices:

  - **Only one node** will fire the job for each firing. This means that, if the job has a repeating trigger which fires every 20 seconds, and its next fire time is at 12:00:00, then only one node will run the job at that time.
  - **Every node** in a cluster will fire the job for each firing, so, if the job has a repeating trigger which fires every 20 seconds, and its next fire time is at 12:00:00, then all the nodes in cluster will run the job at that time.

- Requirements: `com.liferay.portal.scheduler.multiple.jar`, an OSGi bundle which provides scheduler-related clustering features, must be installed and started.

6.1.7 Cache replication

- When Cluster Link is enabled, all the cached values in different nodes of the cluster need to be consistent. This is called cache replication.
- Liferay uses *Ehcache* by default for cache replication.
- In order to improve performance, Liferay provides another tool based on Cluster link as well.

Requirements: `com.liferay.portal.cache.multiple.jar`, an OSGi bundle which provides caching-related cluster features, must be installed and started.

6.1.8 Clustered Document Library Store (I)

Liferay DXP can use the following repositories to store document and media files:

- Case 1: Using the FileSystemStore
  - Uses the file system (local or a mounted share) to store files
- Case 2: Using the AdvancedFileSystemStore
  - Uses the file system (local or a mounted share) to store files
  - Nests the files into more directories by version to store more files and for faster performance

```
dl.store.impl=com.liferay.portal.store.file.system.AdvancedFileSystemStore
```

In both of these cases, the file repository must be accessible from all cluster nodes.

- Case 3: Content Management Interoperability Services (CMIS) store

```
dl.store.impl=com.liferay.portal.store.cmis.CMISStore
```

- Case 4: Use DBStore
  - The document repository is stored in the Liferay database (out of the box with Liferay).

  - Not recommended for performance

```
dl.store.impl=com.liferay.portal.store.db.DBStore
```

- Case 5: **Amazon S3**
  - Stores documents in Amazon's cloud service

```
dl.store.impl=com.liferay.portal.store.s3.S3Store
```

- Case 6: **Other mountable repositories**
  - Documentum
  - Sharepoint repositories
  - Alfresco

Check Marketplace for other repository connectors.

6.1.9 Clustered Document Library Store (II)

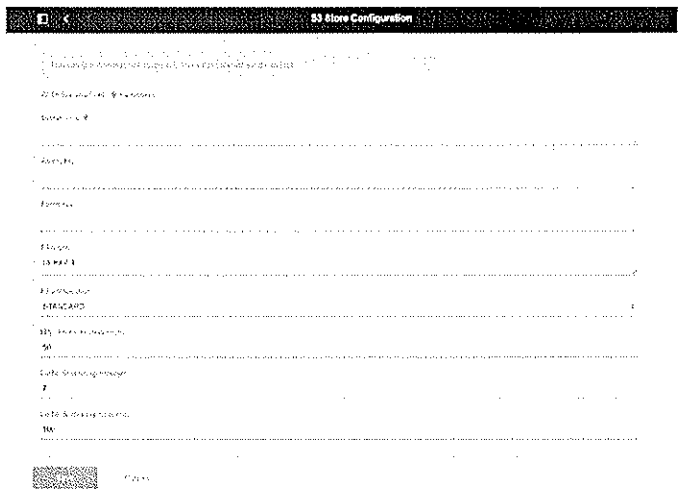
- How to set up Document Library in cluster:
  - **Choose** repository alternatives above - insert `dl.store.impl` into `portal-ext.properties` (all nodes).
  - **Change** store specific attributes in **Control Panel - System Settings** (all nodes).

(Figure 6.2, page 189)

6.1.10 Basic Search Index Clustering

- Default Configuration
  - Embedded ElasticSearch server
  - *Not for production use*
  - *Index replication is not available*
- Recommended Cluster Configuaration
  - Configure Liferay server(s) to use a separate ES server(s).
  - Tune and scale ElasticSearch separately from Liferay.
  - Separated Solr server, Solr server cluster

Figure 6.2:



Workaround for Elasticsearch Server setup for Liferay cluster

- Since Elasticsearch integration is incomplete at the time of writing, we need the following workaround to set up Liferay for the Elasticsearch server:
- **Remove** OSGi bundles from `$LIFERAY_HOME/osgi/portal`.
  - `com.liferay.portal.search.elasticsearch.shield.jar`
  - `com.liferay.portal.search.elasticsearch.shield.fragment.jar`
- **Restart** Liferay.

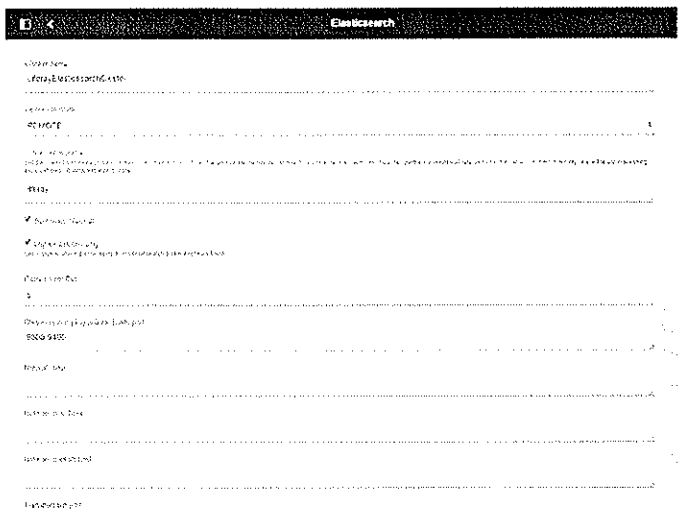
Configuring Elasticsearch Server for Liferay cluster

- UI: Control Panel - System settings - Foundations - Elasticsearch
  - **Set** cluster name.
  - **Change** Operation mode to REMOTE.
  - **Enter** transport address.

(Figure 6.3, page 190)

- Configuration file

Figure 6.3:



  - Create file below in `LIFERAY_HOME/osgi/configs`.
    `com.liferay.portal.search.elasticsearch.configuration.ElasticsearchConfiguration.cfg`

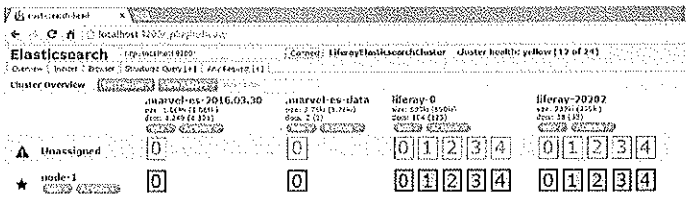
- Set propery values:
  - clustername (name of Elasticsearch server)
  - operationMode (EMBEDDED or REMOTE)
  - transportAddresses (address of remote Elasticsearch cluster)
  - Default ElasticSearch configuration in `config/elasticsearch.yml`
- Trigger reindex after search configuration change (Control Panel - Server Administration - Resources - Reindex all search indexes).

Elasticsearch Server configuration

- Default ElasticSearch configuration in `$ES_HOME/config/elasticsearch.yml`
  - Can set the node and cluster name
- Tip:
  - Set unique names for your ElasticSearch nodes to allow for easier debugging.

- By default, Elasticsearch randomly picks the node name on startup.
- Install **elasticsearch-head** plugin for administrative actions. *(Figure 6.4, page 191)*

Figure 6.4:



6.1.11 HTTPD Server configuration

- Session Replication
- Sync the session across all nodes in your cluster.
  - Pros: User session is not lost if server goes down.
  - Cons: Performance overhead of syncing session across all nodes.
- Server Affinity (or Sticky Sessions)
- Send User requests to same server for entire User session.
  - Pros: Performance – Not syncing session across all nodes
  - Cons: User session is lost if server goes down.

6.1.12 Basic configuration for 2 node cluster

- Configure node 1.
  - **Set** common database.
  - **Enable** cluster link in `portal-ext.properties`.
  - **Set** common Document library repository.
  - **Connect to** separate Elasticsearch server.
- Configure node 2

  - Set common database.
  - Enable cluster link in `portal-ext.properties`.
  - Change default port settings for tomcat (`$TOMCAT_HOME\conf\server.xml`).
  - Change default Gogo shell port (11311) in `portal-ext.properties`: `module.framework.properties.osgi.console=localhost:11311`
  - Set common Document library repository.
  - Connect to separate Elasticsearch server.

- Test nodes.

6.1.13 Advanced configuration for 2 node cluster

- Configure Apache load balancer.
  - Set up load balancer for Liferay nodes (load proxy modules, virtualhost configuration).
- Set up node1 for load balancer.
  - Enable Tomcat clustering (`$TOMCAT_HOME/conf/server.xml`).
  - Prepare Tomcat's worker threads for clustering (`$TOMCAT_HOME/conf/server.xm`
  - Switch on session replication (`$TOMCAT_HOME/conf/web.xml`).
- Set up node2 for load balancer.
  - Apply same steps as per node 1.
- Test cluster.

6.1.14 Cluster troubleshooting

- Liferay provides a built-in feature called debug listener to print verbose logging messages on a console where there are nodes leaving or joining the current cluster.
- It's disabled by default - to enable:
  - Control Panel → Configuration → System Settings → Other → Cluster Executor

Figure 6.5:

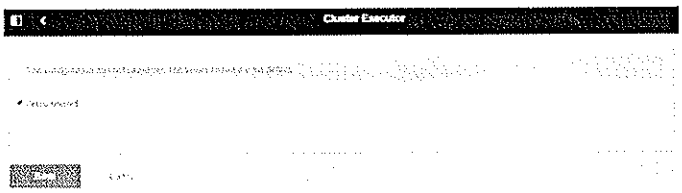


Figure 6.6:

(Figure 6.5, page 193)
(Figure 6.6, page 193)

- Check module statuses with Gogo shell for
  - `com.liferay.portal.cluster.multiple.jar`
  - `com.liferay.portal.scheduler.multiple.jar`
  - `com.liferay.portal.cache.multiple.jar`
- Example:

```
telnet localhost 11311 services | grep com.liferay.
portal.cluster.multiple.jar
```

6.2 Cache, Scheduler and Document repository

- From `CacheStatistics`, you can see the current cache occupancy ratio for specific cache under `Multi-VM / MULTI_VM_PORTAL_CACHE_MANAGER` or `Single-VM / SINGLE_VM_PORTAL_CACHE_MANAGER`
- Examine cache sizes and statistics.

6.2.2 Scheduler

- Quartz is a scheduler that is used within Liferay.
- It has two different modes of operation, using two different sets of database tables.

Features

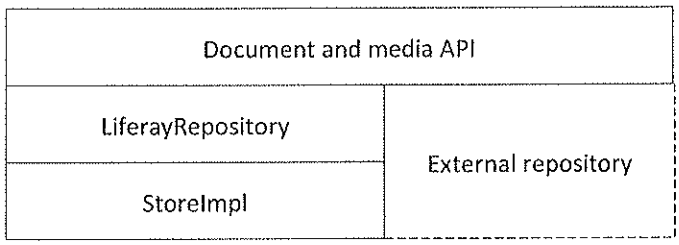
- Quartz is a solution for providing a scalable scheduler for clustered environments.
- Quartz provides for the use of `persistent`, `memory clustered`, and `memory jobs`.
- `Persistent jobs` are jobs stored in the database and will live beyond the life of the containers.
- `Memory clustered jobs` are jobs that are present throughout the cluster, but only held in memory. These are jobs that only execute on a single member of the cluster. An example of this job is the message board email poller. We only want this job to run once within the entire cluster.
- `Memory jobs` are jobs that reside on each instance in the cluster and also execute on each instance of the cluster.

Quartz configuration

Quartz is configured through `portal.properies` and scheduling jobs though indiviual modules.

6.2.3 Document and Media

Figure 6.8:



Architecture

(Figure 6.8, page 197)

The Documents and Media API allows access to the default Liferay repository as well as external repositories. Some examples of external repositories include Sharepoint and Document. New plugins can also be built to connect with external repositories. These external repositories use the same API as Documents and Media. By default, document data is stored in the default Liferay repository.

There are six out of the box store implementations, and common configurations are defined at `portal.properties` under `dl.` prefix.

Store implementations:

- `FileSystemStore` stores files to the file system. With clustering, a shared drive is required.
  - Module: `com.liferay.portal.store.file.system.jar`
    * Configuration UI: `Control Panel` → `Configuration` → `System Settings` → `Foundation` → `Simple File System Store`
    * From configuration file: `{liferay-home}/osgi/configs/com.liferay.portal.store.file.system.configuration.FileSystemStoreConfiguration.cfg`
- `AdvancedFileSystemStore` more efficient than `FileSystemStore` when storing a large amount of files.
  - Module: `com.liferay.portal.store.file.system.jar`
    * Configuration UI : `Control Panel` → `Configuration` → `System Settings` → `Foundation` → `Advanced File System Store`
    * From configuration file:

`{liferay-home}/osgi/configs/com.liferay.portal.store.file.system.configuration.AdvancedFileSystemStoreConfiguration.cfg`

- `DBStore` stores files as BLOBs to Liferay database.
  - Module: `com.liferay.portal.store.db.jar`
    * No configuraiton ui nor configuration file
- `S3Store` uses Amazon S3 as storage
  - Module: `com.liferay.portal.store.s3.jar`
    * Configuration UI : `Control Panel` → `Configuration` → `System Settings` → `Foundation` → `S3 Store`
    * From configuration file: `{liferay-home}/osgi/configs/com.liferay.portal.store.s3.configuration.S3StoreConfiguration.cfg`
- `JCRStore` uses JCR repository as storage.
  - Module: `com.liferay.portal.store.jcr.jar`
    * Configuration UI : `Control Panel` → `Configuration` → `System Settings` → `Foundation` → `JCR Store Configuration`
    * From configuration file: `{liferay-home}/osgi/configs/com.liferay.portal.store.jcr.configuration.JCRStoreConfiguration.cfg`
- `CMISStore` to CMIS compatible interface as storage
  - Module: `com.liferay.portal.store.cmis.jar`
    * Configuration UI : `Control Panel` → `Configuration` → `System Settings` → `Foundation` → `CMIS Store`
    * From configuration file: `{liferay-home}/osgi/configs/com.liferay.portal.store.cmis.configuration.CMISStoreConfiguration.cfg`

6.2.4 External services

Document previews and conversions

- Whenever possible, Liferay generates previews of documents added to the Documents and Media library.

- By default, Liferay uses PDFBox, an open-source Java tool, to generate document previews.
- PDFBox is light-weight and fully-functional, but what if you'd like to enable document previews and conversions for MS Office files, OpenDocument files, audio or video files?
- To accomplish this, Liferay supports integration with several external tools: `ImageMagick`, `OpenOffice` or `LibreOffice`, and `Xuggler`.

ImageMagick

- ImageMagick integration allows Liferay to perform faster and higher-quality document previews.
- In practice, your first step would be to download and install ImageMagick, which is available from http://www.imagemagick.org/.
- For simplicity's sake, we've already installed ImageMagick on your VMs.
- As with each of the three external tools we'll use, ImageMagick can be enabled either in your portal-ext.properties file or via Liferay's control panel.

To enable conversion at Liferay use `portal.properies` or through UI:
`Control Panel` → `Configuration` → `Server Administration` → `External Services` → `Enabling ImageMagick and GhostScript provides document preview functionality.`

Open Office / Libre Office conversions

- Liferay can use Open Office / Libre Office service for the document conversions. To start open Office in server mode you need to use following command.

```
soffice --headless --accept="socket,
host=127.0.0.1,port=8100;urp;" --nofirststartwizard
```

To enable conversion at Liferay use `portal.properies` or through UI:
`Control Panel` → `Configuration` → `Server Administration` → `External Services` → `Enabling OpenOffice integration provides document conversion functionality`

Xuggler

- Xuggler integration enables Liferay to produce audio and video previews. It also lets users play audio and video files within their browsers.
- You can install Xuggler from Liferay's Control Panel by clicking a single button.
- To enable Xuggler, go to `Control Panel` → `Configuration` → `Server Administration` → `External Services` → `Enabling Xuggler provides video conversion functionality`.
- Click `Install` button for eatch node.

6.3 Understanding Search

First, we're going to summarize search implementations in Liferay 6.2 and previous versions. After that, we'll introduce basic concepts of the new search engine, Elasticsearch, and go through the new features of Liferay 7.0 in the search domain.

6.3.1 Search in 6.2 and Previous Versions

1. Lucene 3.5.0 (in 6.2 & 6.1.30): library written (not only) in Java, provide full text search capabilities, (low-level) Java APIs.
2. Liferay Search APIs: **SearchEngine**, **Indexer**, **BaseIndexer**, **IndexSearcher**, **LuceneIndexSearcher**, **BaseSearcher**, **FacetedSearcher**, **IndexAccessor**, **LuceneHelper**, **Query**, **QueryConfig**, **SearchContext**, **Document**, **Field**, etc.
3. Default implementation: **SYSTEM_ENGINE**, **BaseSearchEngine** (aka. Lucene impl)
4. Liferay's core/official plugins code extending/implementing these APIs (e.g. **JournalArticleIndexer**)
5. Search Portlet: provides Faceted Search functionality
6. Search taglibs: *search*, *search_toggle* provide a reusable basic & advanced search box

6.3.2 Extension Points in 6.2 and Previous Versions

- IndexerPostProcessor: defined in **liferay-hook.xml** as `<indexer-post-processor>`
- Custom Indexer: defined in liferay-portlet.xml as `<indexer-class>`

6.3.3 Diving Into Elasticsearch

- Elasticsearch is a highly scalable, distributed, open-source full-text search and analytics engine.
- Open Source
- Store, search & analyze (big volume of) data
- Extensible with plugins, visualization tools
- Default Search Engine in 7.0
- Support Version as of 7.0: 2.2
- Liferay offers additional licenses for ES to provide Level 1-2 support for ES through ES Support.

6.3.4 Elasticsearch Key Features

- (Near)Real-Time Data
- Real-Time Advanced Analytics
- Massively Distributed
- High Availability
- Multitenancy
- Full-Text Search
- Document-oriented
- Schema-Free
- Developer-friendly RESTful API
- Apache 2 Open Source License
- Build on top of Lucene

6.3.5 Elasticsearch Basic Concepts

- Near Real Time (NRT): There is a slight latency (normally one second) from the time you index a document until the time it becomes searchable.
- Cluster: A cluster is a collection of one or more nodes (servers) that together holds your entire data and provides federated indexing and search capabilities across all nodes.
  - Unique name–the default is Elasticsearch.
  - Can be a single node cluster
- Node: A node is a single server that is part of your cluster, stores your data, and participates in the cluster's indexing and search capabilities.
  - Name: a random Marvel character's name by default (default: Elasticsearch)
  - Configure which cluster to join
  - As many nodes as you want
- Index: A collection of documents that have somewhat similar characteristics; e.g., customer data, product catalog, etc.
  - Identified by its name (lowercase!)
  - As many indexes as you want per cluster
  - Lucene index: maximum number of documents in a single index is Integer.MAX_VALUE - 128
- Type: A type is a logical category/partition of your index.
  - In general, a type defined for documents having the same set of common fields
- Document: Basic unit of information that can be indexed; expressed in JSON
  - Must be assigned/indexed to a type in an index

6.3.6 Elasticsearch Scaling

- Elasticsearch provides the ability to subdivide your index into multiple pieces called shards.
  - Number of shards can be defined
  - Each shard is a fully-functional and independent "index" that can be hosted on any node in the cluster.
  - Horizontally split/scale content volume
  - Distribute and parallelize operations across shards (potentially on multiple nodes) → increases performance/throughput
  - Completely controlled by ES → transparent for end-user
- A copy of index's shards → replica shard (or simply replicas)
  - Provides high availability in case of a shard/node fails → shard is never allocated on the same node as the original/primary shard was copied from
  - Allows to scale out search volume/throughput → searches can be executed on all replicas in parallel

6.3.7 Summarizing Shards & Replicas

- Each index can be split into 1 or more shards.
- Each index can be replicated 0/zero or more times → primary shards + replica shards.
- Number of shards & replicas can be defined per index at the time the index is created.
  - Note: After the index is created, you may change the number of replicas dynamically any time, but you cannot change the number shards after-the-fact.
- Default setting:
  - 5 primary shards
  - 1 replica shard

6.3.8 Elasticsearch Installation

Requires Oracle Java 1.7u55+ (1.8u20+ recommended)

1. **Download/Copy** the installation bundle (will be provided).
2. **Ensure JAVA_HOME** is set correctly.
3. Extract to e.g **$LIFERAY_HOME/elasticsearch**.
4. **Go to $LIFERAY_HOME/elasticsearch/bin** and open a terminal (give x-permission to elasticsearch.sh on Linux).
5. **Run** `./elasticsearch` (To run it in the background, add the -d switch to it: $ bin/elasticsearch -d) The default port for REST API is **9200**.

6.3.9 Configuration

- Within the scripts, Elasticsearch comes with built-in JAVA_OPTS passed to the JVM
  - Change -Xmx if necessary.
  - the ES only Java Options can be defined via ES_JAVA_OPTS.
- The ES_HEAP_SIZE will set the same value for both the min and max (to set explicitly: ES_MIN_MEM (default=256m), ES_MAX_MEM (default=1g).
- Make sure to increase the number of open files descriptors on the machine (32k or even 64k recommended).
  - If you want to test the default value, start ES with $ bin/elasticsearch -Des.max-open-files
- Configuration files can be found under $ES_HOME/config folder. (Format YAML)
  - elasticsearch.yml: For configuring ES modules
  - logging.yml: For configuring ES logging, log4j, JSON & properties format are also supported
- Some commonly changed settings:
  - Cluster Name

- Node Name
- Data path
- Network

6.3.10 Elasticsearch Plugins: elasticsearch-head

You may be wondering what you can do to interact with an Elasticsearch cluster. Lucene has Luke, allowing you to view and interact with index data. Elasticsearch has the elasticsearch-head plugin, which provides a web frontend for browsing the Elasticsearch cluster. Running a plugin such as `elasticsearch-head` is the recommended way to interact with an Elasticsearch cluster.

1. `elasticsearch/bin/plugin -install mobz/elasticsearch-head`
2. Open http://localhost:9200/_plugin/head/. It's also possible to run it as a standalone app.

6.3.11 Elasticsearch REST API

Elasticsearch provides a very comprehensive and powerful REST API that you can use to interact with your cluster.

- Check your cluster, node, and index health, status, and statistics.
- Administer your cluster, node, and index data and metadata.
- Perform CRUD (Create, Read, Update, and Delete) and search operations against your indexes.
- Execute advanced search operations such as paging, sorting, filtering, scripting, aggregations, and many other patterns:

```
curl -X <REST Verb> <Node>:<Port>/<Index>/<Type>/<ID>
```

- For example:

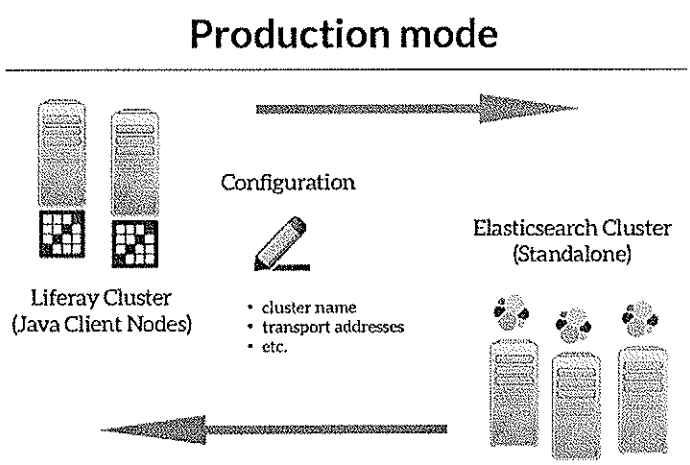
```
curl 'localhost:9200/_cat/indices?v'
curl -XPUT 'localhost:9200/customer?pretty'
curl -XPUT 'localhost:9200/customer/external/1?pretty' \
    -d '{ "name": "John Doe" }'
```

6.3.12 Elasticsearch in Liferay

- Liferay can start an embedded ES node (managed by our **portal-search-elasticsearch** OSGi module). Not supported in Production! Useful for testing.
  - Cluster Name: LiferayElasticsearchCluster
  - Index Home = $LIFERAY_HOME/data/elasticsearch
  - Warning is displayed during startup.
- Remote should be used in real-world scenarios and production deployments.
  - Configuration: → Control Panel → Configuration → System Settings → Platform: Elasticsearch
  - Configuration class: `com.liferay.portal.search.elasticsearch.configuration.ElasticsearchConfiguration`
  - Steps:
    * Set cluster.name: `LiferayElasticsearchCluster` in `$ES_HOME/configs/elasticsearch.xml`

Do a reindex after having changed the configuration. *(Figure 6.9, page 206)*

Figure 6.9:



6.3.13 Elasticsearch in Liferay: Cautions

Liferay 7 configured with Elasticsearch must be configured with Oracle JVM versions 1.7u55+ or 1.8u20+ (as we must fulfill both support matrix): Liferay 7 and Elasticsearch must run with same vendor and version of Java (major and minor version).

- Note: *If your application is written in Java and you are using the transport client or node client, make sure the JVM running your application is identical to the server JVM. In few locations in Elasticsearch, Java's native serialization is used (IP addresses, exceptions, and so forth). Unfortunately, Oracle has been known to change the serialization format between minor releases, leading to strange errors. This happens rarely, but it is best to keep the JVM versions identical between client and server.*

Liferay 7 installed over WebSphere can only be configured with Solr. It cannot be configured with Elasticsearch because:

- IBM JVM is not supported by Elasticsearch.
- Oracle JVM is not supported by WebSphere.

6.3.14 Elasticsearch Plugins: Integration with Liferay

For EE customers, Liferay provides two plugins for Elasticsearch.

Shield

- This is a special module deployed to the platform which will allow the setting of security to the ES server.
- Customers will also need to deploy the Shield ES plugin on the server.

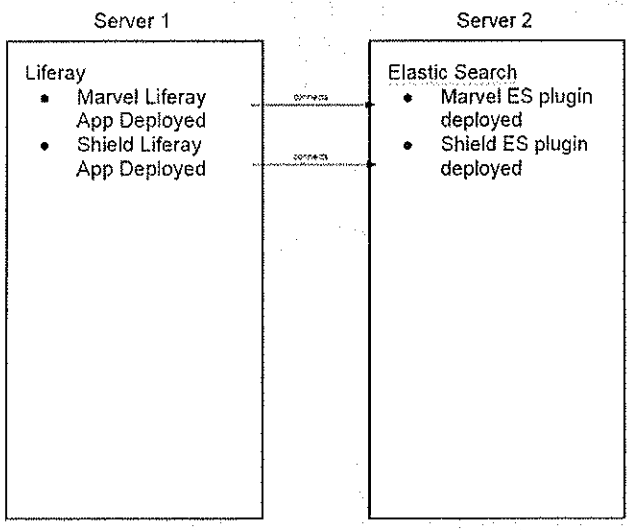
Marvel

- It is necessary to deploy Marvel and Kibana (separate process).
- The module inside the platform allows admins to proxy to Kibana to view the dashboards.

The Liferay Enterprise Search Subscription will look like this: **Gold**: Elasticsearch Support + Marvel **Platinum**: Elasticsearch Support + Marvel + Shield

Customers purchasing an ES license from Liferay will eligible to receive Level1/2 support for Elasticsearch through Liferay's Enterprise Subscription Services. *(Figure 6.10, page 208)*

Figure 6.10:



6.3.15 Conceptual differences between Liferay & Elasticsearch

- As with Solr, it is recommended to install Elasticsearch on a separate server.
- Elasticsearch recommends clusters with odd number of nodes (1, 3, 5 ...).
- Elasticsearch recommends leaving half of the server memory free, in order to be used by operating system cache.

6.3.16 Breaking changes

Lucene libraries and Liferay implementation classes tied to Lucene got removed completely. ES as default Search Engine → removing/stopping the module means you break your search functionality.

6.3.17 portal-search-elasticsearch OSGi module: Overview

- **Configuration**: → Control Panel → Configuration → System Settings → Platform: Elasticsearch
- **Configuration class**: com.liferay.portal.search.elasticsearch.configuration.ElasticsearchConfiguration
- **Runtime**: $LIFERAY_HOME/osgi/portal → com.liferay.portal.search.elasticsearch.jar
- **Source**: ${project.dir}/modules/apps/platform/portal-search/portal-search-elasticsearch

Liferay provides a schema for Elasticsearch to use, inside the OSGi bundle, found in **META-INF/mappings/liferay-type-mappings.json**. *Elasticsearch can operate without a schema, as it tries to determine one based on your data; however, for best results it is recommended to provide one.*

6.3.18 Elasticsearch Configuration File

To connect using a properties file, create a file with the name: `com.liferay.portal.search.elasticsearch.configuration.ElasticsearchConfiguration.cfg` In the osgi/configs folder of your Liferay installation.

Sample configuration file: clusterName=StandAloneElasticSearch operationMode=REMOTE transportAddresses=localhost:9200

6.3.19 New features come with Elasticsearch

- New Queries
  - MatchQuery
  - FuzzyQuery
  - MatchAllQuery

  - MoreLikeThisQuery
  - DisMaxQuery
  - MultiMatchQuery

- New Filters
  - TermsFilter
  - PrefixFilter
  - ExistsFilter
  - MissingFilter
  - QueryFilter
  - Geolocation filters

- New Aggregations
  - Top Hits
  - Extended Stats

6.3.20 IndexerPostProcessor

The **IndexerPostProcessor** allows developers to customize:

- Search queries before they are sent to the search engine
- Documents before they are sent to the search engine
- Summaries for results before they are returned to the end Users; this is the preferred way to customize existing Indexers

To add a new IndexerPostProcessor:

- Implement the interface `com.liferay.portal.kernel.search.IndexerPostProcessor`.
- Publish it to the OSGi registry with the property: indexer.class.name.

6.3.21 Other Customization possibilities

- Adding a new Search Engine Adapter
- Customizing IndexerRequestBufferOverflowHandler
- Customizing HitsProcessors

6.4 Solr (Optional)

Besides the default Elasticsearch, Liferay supports Solr as an alternative search engine.

- Standalone enterprise search server based on Lucene
- Highly reliable, scalable, and fault tolerant
- Provides distributed indexing, replication, and load balanced querying
- Supports automatic failover and recovery

The latest Solr version is 5.5.0.

6.4.1 Solr configuration

Exercise: Solr server setup (I)

Please note that Solr requires Java 1.7u55 or later to run properly.

1. Unzip `solr-5.2.1.zip` from `~/Downloads` into a local directory (Alternatively download from http://archive.apache.org/dist/lucene/solr/5.2.1/solr-5.2.1.zip)
2. Set SOLR_HOME system variable to point to

   `[SOLR_INSTALLATION]/solr-5.2.1/server/solr`
3. Create folder called `liferay` inside `$SOLR_HOME`
4. Create 2 subdirectories in the `liferay` folder.
   - `conf`
   - `data`
5. Copy the content of `$SOLR_HOME/configsets/data_driven_schema_configs/conf` to `$SOLR_HOME/liferay/conf`.

Exercise: Solr server setup (II)

1. Extract `solrconfig.xml`, `schema.xml` from Liferay Solr OSGI module (`com.liferay.portal.search.solr.jar`).
2. Copy `solrconfig.xml`, `schema.xml` files to `$SOLR_HOME/liferay/conf/`.
3. Because of LPS-64274 exceptions can be thrown due to some missing mapping in our `schema.xml`.

   Workaround: modify the field definition of configurationModelAttributeName in the `schema.xml`:

   ```
   <field indexed="true" name="configurationModelAttributeName"
   multiValued="true" stored="true" type="string" />
   ```

Save modified file.

1. Create `core.properties` file in `SOLR_HOME/liferay` with content.

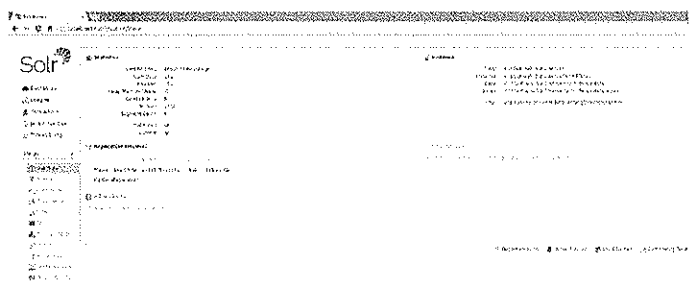
   ```
   name=liferay
   config=solrconfig.xml
   schema=schema.xml
   dataDir=data
   ```

2. Start the Solr server in `$SOLR_HOME/../../bin ./solr start -f`.

   Solr server will listen on port 8983 by default.

3. Go to `http://localhost:8983/solr/#/~cores` and check server is up and running. *(Figure 6.11, page 212)*

Figure 6.11:



Configure Liferay for Solr

1. **Make sure** that platform is down
2. **Delete** Elasticsearch modules from `$LIFERAY_HOME\osgi\portal`

   ```
   com.liferay.portal.search.elasticsearch.jar
   com.liferay.portal.search.elasticsearch.shield.fragment.jar
   com.liferay.portal.search.elasticsearch.shield.jar
   ```

3. **Copy** `com.liferay.portal.search.solr.jar` into `$LIFERAY_HOME\osgi\portal`.
4. **Create** new Solr configuration file in `$LIFERAY_HOME\osgi\configs` named `com.liferay.portal.search.solr.configuration.SolrConfiguration-default.cfg`
5. **Add** parameters to file.

   ```
   readURL=http://localhost:8983/solr/liferay
   writeURL=http://localhost:8983/solr/liferay
   ```

6. **Start** Liferay.
7. **Check** Solr admin console to see whether or not indexing is working correctly.

Solr as OSGi module

Runtime: `$LIFERAY_HOME/osgi/portal` → `com.liferay.portal.search.solr.jar`
Configuration: Control Panel → Configuration → System Settings → Foundation

- SOLR *(Figure 6.12, page 214)*
- SOLR HTTP Client Factory *(Figure 6.13, page 214)*
- SOLR SSL Factory *(Figure 6.14, page 214)*

Upgrade considerations

- Upgrading from previous Liferay versions using Solr (6.2 or earlier) is supported.
- Required steps:

Figure 6.12:

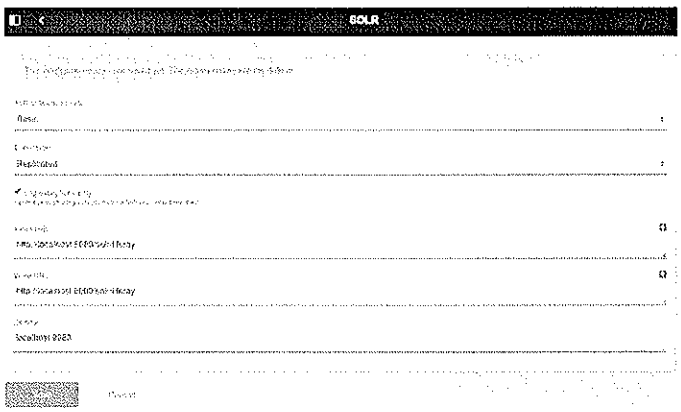


Figure 6.13:

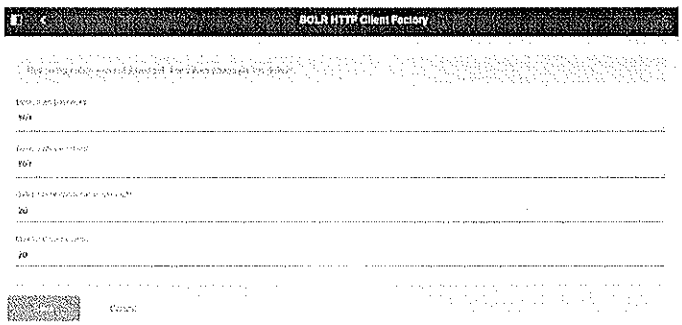


Figure 6.14: