Liferay Digital Experience Platform 7.2

Deployment Checklist

Table of Contents

Introduction

The Liferay Engineering and Global Services teams have performed intensive performance and scalability testing on Liferay Digital Experience Platform (DXP) and the accumulated knowledge has been condensed into this checklist. Although a Liferay DXP deployment's performance profile may differ depending on several factors, this checklist provides a list of critical parameters to monitor for tuning and some initial settings to use as a starting point. These initial settings have withstood heavy testing by the Liferay Engineering scalability team.

The Liferay Global Services team also has a specialized Go Live package that can help with your pre-production tuning and configuration.

Reference Architecture

The selection of an appropriate architecture is one of the first decisions in your deployment path. To select an appropriate architecture, you must consider:

- Information Security: Securing sensitive hardware and information from malicious attack and intrusion.
- Performance: Supporting the desired number of total users, concurrent transactions, etc.
- Fault Tolerance: Maintaining uptime during unexpected failure or scheduled maintenance.
- Flexibility and Scalability: Designing an expandable architecture to support additional features and users without significant redesign.

Although appearing somewhat complex, the reference architecture depicted in Figure 1 provides high levels of fault tolerance and flexibility.

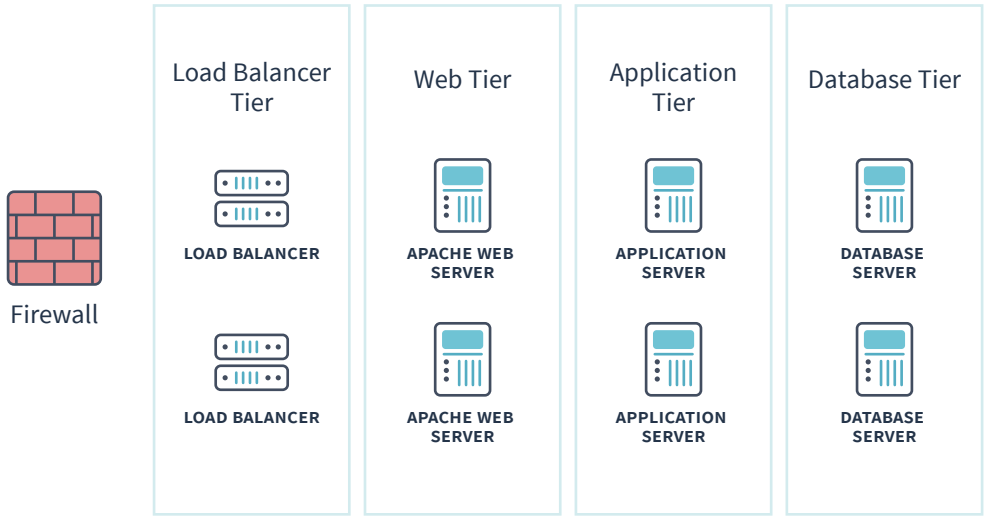

Figure 1 - Liferay DXP Reference Architecture

The architecture contains the following tiers:

- **Firewall:** Intrusion detection and prevention.
- **Load Balancer Tier:** Ensures smooth distribution of load between multiple web server resources.
- **Web Server Tier:** Delivers static content elements like images, rich media, CSS files, etc. Also provides integration modules to single sign-on solutions like CA Siteminder, Oracle Identity, Ping, etc. Keep in mind a content delivery network (CDN) may take some of static file caching and edge caching the responsibilities of the web server. However, you may still need this layer to more easily achieve functions like URL rewriting.
- **Application Tier:** Hosts Liferay supported application servers like Tomcat, JBoss, Oracle Weblogic, and IBM Websphere (please see Liferay DXP Compatibility Matrix for additional application servers). Also hosts search engines like Solr and Elasticsearch.
- **Database Tier:** Hosts Liferay supported database servers like MySQL, Oracle, MS SQL, IBM DB2, Postgres (please see Liferay DXP Compatibility Matrix for additional platforms).

The hardware deployed within each tier varies depending on the type of transactions. We will use Liferay Engineering's benchmarking environment as a hardware specification guide:

LOAD BALANCER TIER

Cisco Load Director or Cisco Content Services Switch (CSS) or F5 Big-IP

WEB TIER

Provides caching, compression and other capabilities using Apache, Nginx, Varnish, etc.

- 1 - Intel Core 2 Duo E6405 2.13GHz CPU, 4GB memory, 1-146GB 10k RPM SCSI

APPLICATION TIER

Represents the workhorse of the architecture

- 2 - Intel Xeon E5-2643 v4 3.40GHz CPU, 64GB memory, 2-300GB 15K RPM SATA 6Gbps - used for Liferay Portal
- 2- Intel Xeon E5-2643 v4 3.40GHz CPU, 64GB memory, 2-300GB 15K RPM SATA 6Gbps - used for Elasticsearch

DATABASE TIER

- 2 - Intel Xeon E5-2643 v4 3.40GHz CPU, 64GB memory, 2-300GB 15K RPM SATA 6Gbps

Although the application servers have 64GB of physical memory, you may choose to deploy with less if your Java Virtual Machine (JVM) does not utilize large heap sizes. Modern operating systems will also use any available physical memory for file system caches.

Virtualized and Cloud Deployments

While the reference architecture describes a physical deployment, the same concepts may be applied to a cloud based or virtualized deployment. Many Liferay customers choose to deploy on either public clouds (e.g., Amazon EC2) or their own private clouds (e.g., VMWare VSX based private cloud). Each physical machine may be replaced by appropriate quantities of virtual machines. Additionally, customers can use Liferay DXP Cloud, our enterprise PaaS, to manage and deploy Liferay DXP. Each physical machine may be replaced by appropriate quantities of virtual machines.

In the virtualized deployments, it is critical to allocate sufficient CPU resources. For instance, for systems deployed to Amazon AWS, allocated CPUs are calculated using Amazon EC2 Compute Units. However, 1 Compute Unit does not equal to 1 physical CPU or even 1 core on a CPU. In Amazon's terms, each application server used in the reference architecture equates to roughly a "Cluster Compute Quadruple Extra Large Instance," or 33.5 EC2 Compute Units. Thus, to properly plan the virtualized/cloud deployment, customers must account for not only virtualization overhead, but also ensure allocation of sufficient CPU resources.

Fault Tolerance

The reference architecture is fault tolerant at every level. With clusters at the web, application and database tier, you may suffer a catastrophic hardware failure of any node and continue to service users with little performance degradation.

The depicted reference architecture represents the minimum deployment units to ensure proper fault tolerance within a single data center. You may increase the number of servers within each tier according to your load patterns to achieve a multiplier effect in the number of users the deployment can support while maintaining sufficient fault tolerance.

Multi-data-center fault tolerant architectures are not provided as part of the reference architecture.

Performance

Each deployment's performance characteristics will vary depending on the type of activity and the performance of custom application elements. Liferay Engineering has created a series of scenarios to benchmark Liferay DXP's out-of-the-box performance characteristics for content management, collaboration and social enterprise scenarios. Results from these reference architectures have indicated Liferay DXP can support over 47,000 virtual collaboration users and over 79,000 logins per minute with average login times under 312 milliseconds. Liferay DXP accomplished this load within the reference architecture while utilizing no more than 40 percent of CPU resources in the Web Tier, 95 percent of CPU resources in the Application Tier and less than 5 percent of CPU resources in the Database Tier.

Scalability

Liferay Engineering's testing has shown Liferay DXP to scale linearly. Thus, if you know a single application server supports X virtual users and assuming sufficient database and web server resources, you may calculate the total number of application servers required.

Security

The firewall preceding the Load Balancer Tier will provide sufficient intrusion detection and prevention. However, depending on your organization's information security requirements, you may introduce additional firewall layers between each tier to further secure the infrastructure.

Liferay DXP Tuning Guidelines

When tuning your DXP, there are several factors to take into consideration, some specific to Liferay DXP, while others are concepts that apply to all Java and Java enterprise applications. The following guidelines are meant to serve as an initial baseline from which to tune your specific deployment.

Application Server Tuning

Although the actual setting names may differ, the concepts are applicable across most application servers. We will use Tomcat as an example to demonstrate application server tuning concepts. You should also consult your application server provider's documentation for additional specific settings that they may advise.

Database Connection Pool

The database connection pool is generally sized at roughly 30 to 40 percent of the thread pool size. The connection pool provides a connection whenever DXP needs to retrieve data from the database (e.g., user login, etc). If this size is too small, requests will queue in the server waiting for database connections. However, too large a setting will mean wasting resources with idle database connections.

As with thread pools, you should monitor these settings and adjust them based on your performance tests.

TOMCAT 9.0.X

In Tomcat, the connection pools are configured in the Resource elements in $CATALINA_HOME/conf/ Catalina/localhost/ROOT.xml.

```
<Resource name="jdbc/LiferayPool" auth="Container"
      factory="com.zaxxer.hikari.HikariJNDIFactory"
      type="javax.sql.DataSource"
      minimumIdle="10"
      maxLifetime="0"
      maximumPoolSize="85"
      driverClassName="com.mysql.jdbc.Driver"
      dataSource.user="XXXXXX"
      dataSource.password="XXXXXXXXX"
jdbcUrl="jdbc:mysql://localhost/lportal?characterEncoding=UTF-
8&amp;dontTrackOpenResources=true&amp;holdResultsOpenOverStateme
ntClose=true&amp;useFastDateParsing=false&amp;useUnicode=true"
/>
```

In this configuration, we start with 10 connections and a maximum of 85 connections in the pool.

You may choose from a variety of database connection pool providers, including C3P0, HikariCP and Tomcat. You may also choose to configure the Liferay JDBC settings in your portal.properties. The choice of JNDI or portal.properties should be something you do based on your application server choice and IT standards. For instance, if you are using Tomcat and your internal standards do not dictate JNDI for JEE resource declarations, then using portal-ext.properties is the simplest path.

Deactivate Development Settings in the JSP Engine

Most application servers have their JSP Engine configured for development mode. Liferay recommends deactivating many of these settings prior to entering production:

- Development mode: This will enable the JSP container to poll the file system for changes to JSP files.
- Mapped File: Generates static content with one print statement versus one statement per line of JSP text.

TOMCAT 9.0.X

In the $CATALINA_HOME/conf/web.xml, update the JSP servlet to look like the following:

```
<servlet>
      <servlet-name>jsp</servlet-name>
      <servlet-class>org.apache.jasper.
servlet.JspServlet</servlet-class>
      <init-param>
            <param-name>development</param-name>
            <param-value>false</param-value>
      </init-param>
      <init-param>
            <param-name>mappedFile</param-name>
            <param-value>false</param-value>
      </init-param>
      <load-on-startup>3</load-on-startup>
</servlet>
```

Thread Pool

Each incoming request to the application server consumes a worker thread for the duration of the request. When no threads are available to process requests, the request will be queued waiting for the next available worker thread. In a finely tuned system, the number of threads in the thread pool should be relatively balanced with the total number of concurrent requests. There should not be a significant amount of threads left idle waiting to service requests.

Liferay Engineering recommends setting this initially to 50 threads and then monitoring it within your application server's monitoring consoles. You may wish to use a higher number (e.g., 250) if your average page times are in the 2 to 3s range. Too few threads in the thread pool may lead to excessive request queuing while too many threads may lead to excessive context switching.

TOMCAT 9.0.X

In Tomcat, the thread pools are configured in the Connector element in $CATALINA_HOME/conf/server.xml. Further information can be found in the Apache Tomcat documentation. Liferay Engineering used the following configuration during testing:

```
<Connector maxThreads="75" minSpareThreads="50"
maxConnections="16384"
port="8080" connectionTimeout="600000"
redirectPort="8443" URIEncoding="UTF-8"
maxKeepAliveRequests="-1" address="xxx.xxx.xxx.xxx"/>
```

Additional tuning parameters around Connectors are available, including connector types, connection timeouts and TCP queue. You should consult the appropriate Tomcat documentation for further details.

Java Virtual Machine Tuning

Tuning the JVM primarily focuses on tuning the garbage collector and the Java memory heap. These parameters look to optimize the throughput of your application. We used Oracle's 1.8 JVM for the reference architecture. You may also choose other supported JVM versions and implementations. Please consult the Liferay DXP Compatibility Matrix for additional compatible JVMs.

Garbage Collector

Choosing the appropriate garbage collector (GC) will help improve the responsiveness of your Liferay DXP deployment. Liferay recommends using the concurrent low pause collectors:

```
-XX:+UseParNewGC -XX:ParallelGCThreads=16 -XX:+UseConcMarkSweepGC
-XX:+CMSParallelRemarkEnabled -XX:+CMSCompactWhenClearAllSoftRefs
-XX:CMSInitiatingOccupancyFraction=85 -XX:+CMSScavengeBeforeRemark
```

You may choose from other available GC algorithms including parallel throughput collectors and G1 collectors. Liferay recommends first starting your tuning using parallel collectors in the new generation and concurrent mark sweep (CMS) in the old generation.

Note: The value 16 in "ParallelGCThreads=16" will vary based on the type of CPUs available. We recommend setting the value according to CPU specification. On Linux machines, you may find the number of available CPUs by running "cat /proc/cpuinfo".

Note: There are additional "new" algorithms like G1, but Liferay Engineering's tests for G1 have indicated that it does not improve performance. Your application performance may vary and you should add it to your testing and tuning plans.

Java Heap

When most people think about tuning the Java memory heap, they think of setting the maximum and minimum memory of the heap. Unfortunately, most deployments require far more sophisticated heap tuning to obtain optimal performance, including tuning the young generation size, tenuring durations, survivor spaces and many other JVM internals.

For most systems, Liferay recommends starting with at least the following memory settings:

```
-server -XX:NewSize=1024m -XX:MaxNewSize=1024m -Xms4096m -Xmx4096m
-XX:MetaspaceSize=512m -XX:MaxMetaspaceSize=512m
-XX:SurvivorRatio=6
-XX:TargetSurvivorRatio=90 -XX:MaxTenuringThreshold=15
```

On systems that require large heap sizes (e.g., above 4GB), it may be beneficial to use large page sizes. You may activate large page sizes using the following JVM options:

```
-XX:+UseLargePages -XX:LargePageSizeInBytes=256m
```

You may choose to specify different page sizes based on your application profile.

Note: To use large pages in the JVM, you must configure your underlying operating system to activate them. In Linux, run "cat /proc/meminfo" and look at "huge page" items.

Caution: You should avoid allocating more than 32GB to your JVM heap. Your heap size should be commensurate with the speed and quantity of available CPU resources.

JVM Advanced Options

The following advanced JVM options were also applied in the Liferay benchmark environment:

```
-XX:+UseCompressedOops -XX:+DisableExplicitGC
```

Please consult your JVM documentation for additional details on advanced JVM options.

Combining the above parameters together, we have:

```
-server -XX:NewSize=1024m -XX:MaxNewSize=1024m -Xms4096m -Xmx4096m
-XX:MetaspaceSize=512m -XX:MaxMetaspaceSize=512m
-XX:SurvivorRatio=6
-XX:TargetSurvivorRatio=90 -XX:MaxTenuringThreshold=15
-XX:+UseLargePages
-XX:LargePageSizeInBytes=256m -XX:+UseParNewGC
-XX:ParallelGCThreads=16
-XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled
-XX:+CMSCompactWhenClearAllSoftRefs
-XX:CMSInitiatingOccupancyFraction=85
-XX:+CMSScavengeBeforeRemark -XX:+UseCompressedOops
-XX:+DisableExplicitGC
```

Caution: The above JVM settings should formulate a starting point for your performance tuning. Each system's final parameters will vary due to a variety of factors including number of current users and transaction speed.

Liferay recommends monitoring the garbage collector statistics to ensure your environment has sufficient allocations for metaspace and also for the survivor spaces. Simply using the guideline numbers above may result in dangerous runtime scenarios like out of memory failures. Improperly tuned survivor spaces also lead to wasted heap space.

Monitoring GC and JVM

Although the previously introduced parameters give you a good start to tuning your JVM, you must monitor GC performance to ensure you have the best settings to meet your needs. There are several tools to help you monitor Oracle JVM performance including:

VISUAL VM

This tool provides a centralized console for viewing Oracle JVM performance information, including garbage collector activities.

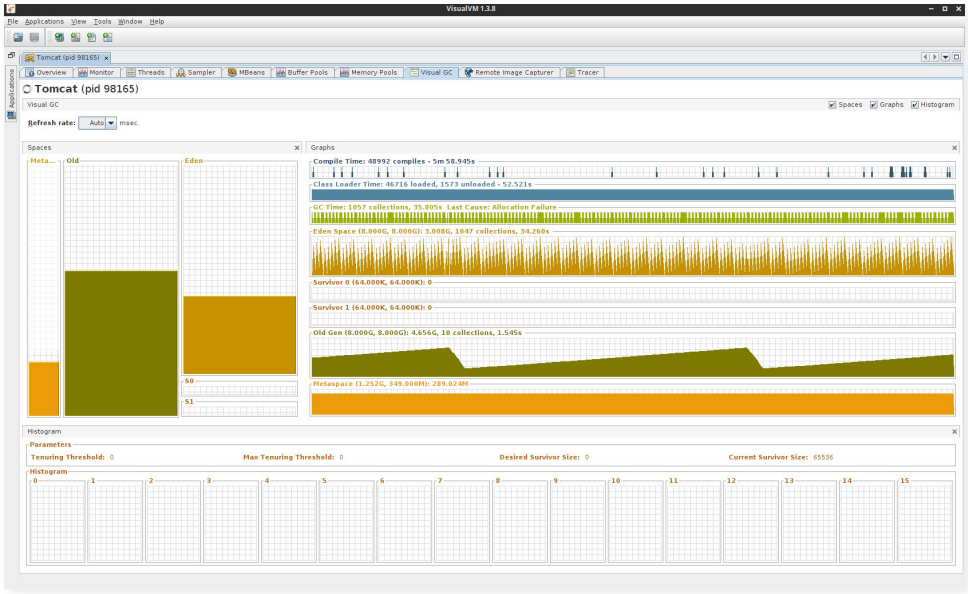

Figure 2 - Visual VM's Visual GC

JMX CONSOLE

This tool helps display various statistics like Liferay's distributed cache performance, the performance of application server threads, JDBC connection pool usage, etc.[1]

Add the following to your application server's JVM arguments to enable JMX connections:

```
-Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.port=5000
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

1 The JMX Console is the preferred tool to use when observing Tomcat performance information.

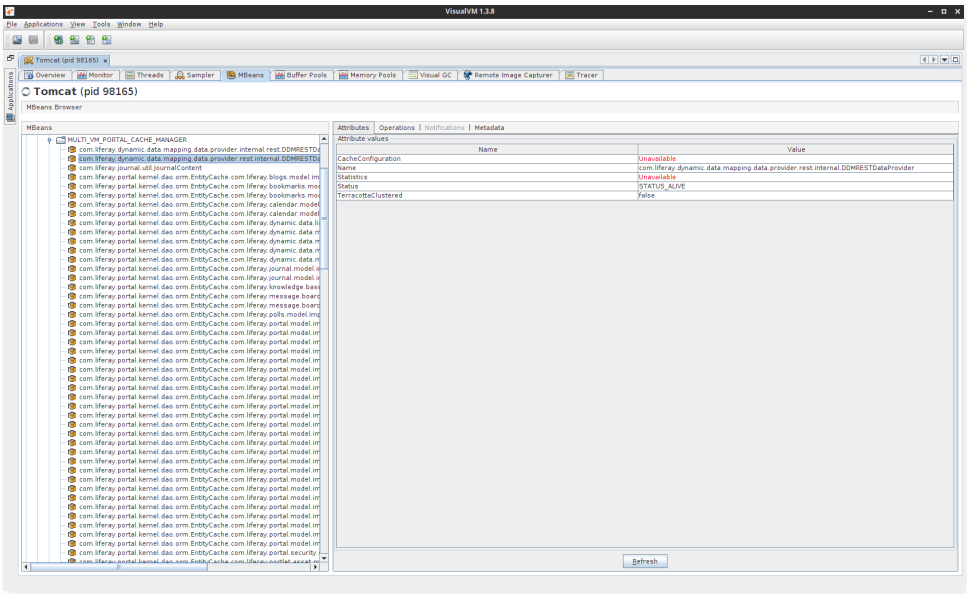

Figure 3 - Visual VM's JMX Console

GARBAGE COLLECTOR VERBOSE LOGGING

Add the following to your JVM arguments to activate verbose logging for the JVM garbage collector:

```
-verbose:gc -Xloggc:/tmp/liferaygc1.log -XX:+PrintGCDetails
-XX:+PrintGCCause -XX:+PrintGCApplicationConcurrentTime
-XX:+PrintGCApplicationStoppedTime
```

You will need these logs to properly tune the JVM.

Note: To ensure you do have sufficient debugging information should your JVM encounter out of memory scenarios, you should consider adding:

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/tmp/dumps
```

Liferay DXP Tuning Parameters

Liferay DXP ships with many parameters already optimized for performance. However, depending on your specific use case, you may wish to tune additional settings. Unless otherwise stated, you may tune these parameters in the portal.properties file[2] or using System Settings.

Caching

> **WARNING:** Care should be taken when tuning caching. The size of the cache will have a direct impact on the amount of memory your application server has to work with. The larger the cache, the less memory available for processing requests.
>
> If your application caching requirements exceed what the out of the box, in-process provides in scalability, Liferay recommends investigating the use of Terracotta as a clustered, out of process cache.

Liferay DXP relies upon both content and object caching to minimize database interaction and excessive object creation. Out of the box, Liferay DXP leverages EHCache for its caching needs. You may configure additional caches including Terracotta, Oracle Coherence Cache, etc. For the purposes of our discussion, we will focus on the out-of-box EHCache.

To monitor the caches, you will need to rely upon the JMX Console.[3] In the preceding figure, we see the JMX Console view for monitoring a cache used to store user information:

CacheHits: Displays the number of requests that successfully retrieved from the cache rather than going to the database. This includes objects stored in memory and stored on disk.

CacheMisses: Displays the number of requests that could not find its object in cache and thus had to retrieve from the database.

2 We recommend that you consult the official Liferay DXP documentation for instructions on how to override properties stored in portal.properties, how to change settings using System Settings and how to export and import settings using System Settings.

3 You may also use Liferay Connected Services to help monitor cache utilization.

InMemoryHits: Displays the number of requests that successfully retrieved from the in-memory cache. This does not include requests that triggered retrievals from on disk storage.

ObjectCount: The total number of objects in cache.

OnDiskHits: The total number of requests that could not find their objects in memory, but successfully located the objects on the local file system. This is only applicable if you have enabled cache overflow to disk.[4]

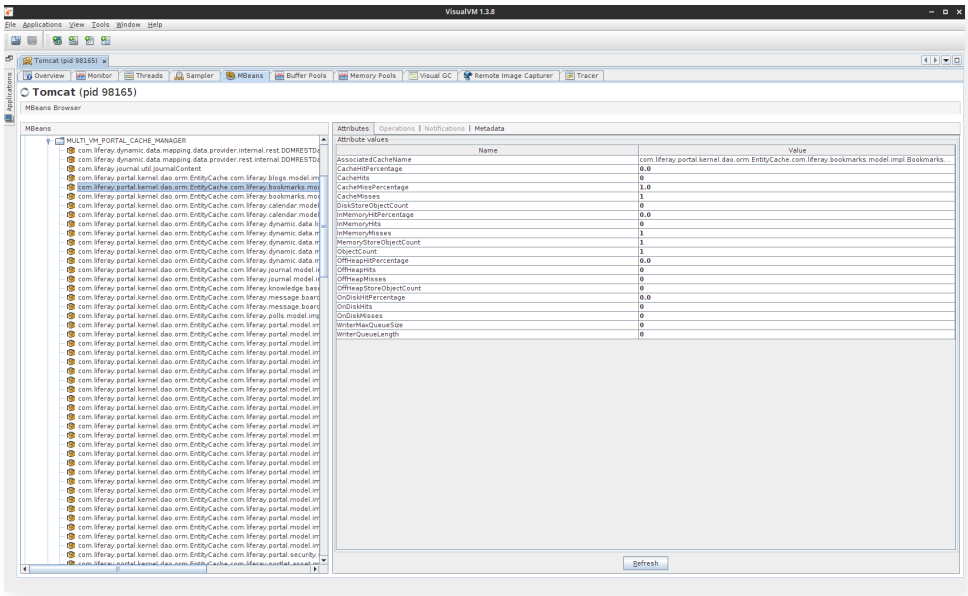

Figure 4 - Monitoring Liferay Data Caches

By default, Liferay DXP's cache configuration uses the memory store and allows for a maximum of 10,000 elements. Although using a disk store may increase the size of the cache, Liferay does not recommend this approach due to the increased dependency upon disk IO operations.

Let us use the user cache as our tuning example. Monitoring provides the number of times an object was returned from the cache. On the other hand, if the object wasn't cached, it will return a miss when performing a search. Usually you can determine whether the user cache will require tuning in order to improve the rate of an object being returned, due to the number of cache misses.

4 Cache overflow to disk means allow Ehcache to serialize (write) objects to the local file system. This is akin to using virtual memory page at the operating system level.

You may tune and override Liferay's cache settings either via a plugin or via changes to portal-ext.properties.

Basic Liferay DXP data caches of interest include:

```
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portal.model.impl.AccountImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portal.model.impl.ContactImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portal.model.impl.GroupImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portal.model.impl.LayoutImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portal.model.impl.LayoutSetImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portal.model.impl.ResourceImpl
```

The following caches may be of interest, depending on the features you are leveraging in the product:

```
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portlet.blogs.model.impl.BlogsEntryImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.
liferay.portlet.wiki.model.impl.WikiPageImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.liferay.
portlet.messageboards.model.impl.MBCategoryImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.liferay.
portlet.messageboards.model.impl.MBThreadImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.liferay.
portlet.journal.model.impl.JournalArticleImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.liferay.
portlet.journal.model.impl.JournalStructureImpl
com.liferay.portal.kernel.dao.orm.EntityCache#com.liferay.
portlet.journal.model.impl.JournalTemplateImpl
```

CACHE REPLICATION

Liferay DXP ships with an enhanced algorithm that uses more efficient thread pooling for cached object and event replication. To configure the DXP to use this algorithm, simply activate clustering using the following portal.properties:

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

COUNTER INCREMENT

Liferay DXP uses an internal sequence generator for generating object IDs. Increasing the increment size of this counter will reduce the number of times it must communicate with the database to reserve IDs. We recommend setting this number at roughly 2000, depending on the load of your system. You may do so by adding the following to portal-ext.properties:

```
counter.increment=2000
```

DOCUMENT LIBRARY PREVIEWS

Liferay DXP's Document Library's preview capabilities enable users to view assets stored in the repository without downloading the file. The document preview feature relies upon PDFBox and OpenOffice Server or ImageMagick.

The PDFBox and OpenOffice Server approach uses OpenOffice Server to first convert documents to PDF and then uses PDFBox to generate images from the PDFs. This approach may be less accurate and certain types of files may not render properly. To configure this approach, please set the following properties in your portal-ext.properties:

```
openoffice.server.enabled=true
openoffice.server.host=<IP_ADDRESS>
openoffice.server.port=<PORT>
openoffice.cache.enabled=true
```

You must also install OpenOffice in "server" or "headless" mode.

The ImageMagick approach requires installation of ImageMagick on your OS hosting Liferay DXP. You may obtain ImageMagick from imagemagick.org. To configure DXP to use ImageMagick, you must configure:

```
imagemagick.enabled=true
imagemagick.global.search.path[apple]=/opt/local/bin:/opt/local/
share/ghostscript/fonts:/opt/local/share/fonts/urw-fonts
```

```
imagemagick.global.search.path[unix]=/usr/local/bin:/usr/local/
share/ghostscript/fonts:/usr/local/share/fonts/urw-fonts
imagemagick.global.search.path[windows]=C:\\Program
Files\\gs\\bin;C:\\Program Files\\ImageMagick
```

You may wish to further tune ImageMagick depending on your document sizes and other parameters. You may find more details at: imagemagick.org/script/architecture.php.

In addition, preview generation can be quite expensive. Thus, by default, Liferay DXP executes preview generation in a separate JVM:

```
dl.file.entry.preview.fork.process.enabled=true
```

This helps improve stability for the DXP Java Virtual Machine.

Document Library Storage

Liferay DXP's Document Library can be configured with a variety of storage engines including Amazon S3, database and file system.

Liferay provides two file system based storage facilities: FileSystemStore and AdvancedFileSystemStore. The FileSystemStore stores files using a path structure of *${liferay.home}/document_library/<companyId>/<groupId>/<fileName>/<version>*. The AdvancedFileSystemStore stores files using a path structure of *${liferay.home}/document_library/<companyId>/<groupId>/<fileName_with_extension>/<fileName_without_extension>_<version>/<version>*. The additional directory hierarchy will help improve performance and scalability. Consequently, Liferay recommends using the AdvancedFileSystemStore for production use.

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

It is important to note that Liferay does not handle file system backups, replication and file locking. Liferay DXP relies upon your selected OS, backup and file system replication tools for those facilities.

If you do not have file system replication or backup capabilities, then the DBStore may be a better choice. DBStore stores the asset binaries into a relational database. This allows you to use the database's replication and backup facilities to also backup your document assets. However, you will experience slower download performance. This option is not recommended for Liferay DXP deployments with large amounts of document downloads.

Liferay also provides JCR and CMIS adapters for storage. These are not recommended for production use.

Direct Servlet Context Reload

In many production systems, we do not expect to need the servlet and JSP containers to reload JSPs, because there will be no administrative operations that require JSP dynamic reloading at runtime, such as:

- Manual stop / start of a module that contains a JSP file.
- Deploy a new version of an existing OSGi module.
- Deploy a new fragment to override a JSP file of an existing OSGi module.

Consequently, in production environments where we can assume that none of the above occur, we should be able to set the following value in portal-ext.properties:

```
direct.servlet.context.reload=false
```

Enabled Locales

Liferay DXP is capable of supporting many different languages. You may wish to reduce the number of available languages in your solution. The fully supported locales can be found in the following property:

```
locales.enabled=ca_ES,zh_CN,nl_NL,en_US,fi_FI,fr_FR,
de_DE,iw_IL,hu_HU,ja_JP,pt_BR,es_ES
```

There are also languages that are currently undergoing translation and thus under beta support:

```
locales.beta=ar_SA,eu_ES,bg_BG,ca_AD,zh_TW,hr_HR,cs_CZ,da_
DK,nl_BE,en_GB,en_AU,et_EE,gl_ES,el_GR,hi_IN,in_ID,it_
IT,ko_KR,lo_LA,lt_LT,nb_NO,fa_IR,pl_PL,pt_PT,ro_RO,ru_
RU,sr_RS,sr_RS_latin,sl_SI,sk_SK,sv_SE,tr_TR,uk_UA,vi_VN
```

To use these locales, simply add them to locales.enabled in your portal-ext.properties. For improved user experience for your content managers, you may wish to reduce the locales.enabled to only those that your solution needs to support. You can add more locales in the future by simply adding to the locales.enabled property.

Encryption algorithms

If you intend to use Liferay's out-of-the-box login facilities and store user passwords in the database, you may wish to tune the encryption algorithm used for passwords. By default, the value is:

```
passwords.encryption.algorithm=PBKDF2WithHmacSHA1/160/128000
```

It is a Password-Based Key Derivation Function algorithm that generates a 160 bit hash after 128000 encryption rounds (2014 OWASP recommendations).

However, in deployments less powerful CPUs (e.g., vCPUs or older CPUs), you may wish to use a different algorithm or reduce the number of rounds. For instance, you can reduce the number of encryption rounds. Fewer rounds of encryption will improve performance by reducing CPU usage. You should consult your information security organization for proper guidance on password hashing guidelines.

Groups Complex SQL

Liferay DXP uses a Group object to help track certain entities that require site-like features. For instance, organizations with sites, users with private and public profile pages, etc. The number of groups can become quite large and complex as the number of organizations and users grow. Liferay has implemented two ways to optimize querying for these groups:

1. Complex SQL which will increase utilization on the database side or
2. More in-memory processing which will increase memory consumption and CPU usage on the DXP JVM.

Option one is suitable when you have a large number of groups of a particular type (e.g., users) while option two will help reduce database CPU usage but increase CPU and memory usage on the DXP JVM. Thus, there is a tradeoff decision to be made.

To configure additional types for complex SQL, you may update groups.complex.sql.class.names in portal-ext.properties. The default only includes com.liferay.portal.kernel.model.User. You may wish to include other available models including:

1. `com.liferay.portal.kernel.model.Organization`: If you have a large number of organizations.
2. `com.liferay.portal.kernel.model.UserGroup`: If you have a large number of user groups.
3. `com.liferay.portal.kernel.model.LayoutPrototype and com.liferay.portal.kernel.model.LayoutSetPrototype`: If you are using staging with Layout versions and/or branching.

For instance:

```
groups.complex.sql.class.names=com.liferay.portal.kernel.
model.User,com.liferay.portal.kernel.model.UserGroup
```

Message Bus

> **WARNING:** This is an advanced configuration and should be modified sparingly. The Liferay Message Bus has been tuned to handle most scenarios.

Liferay DXP leverages asynchronous messaging, via the Liferay Message Bus, for many of its services like mail and indexing. The bus provides a loosely coupled, pluggable architecture that helps improve user experience by performing system tasks (e.g., email notifications) in the background.

Liferay recommends monitoring and tuning the message bus according to your use case. You can monitor message bus statistics via the JMX Console discussed previously.

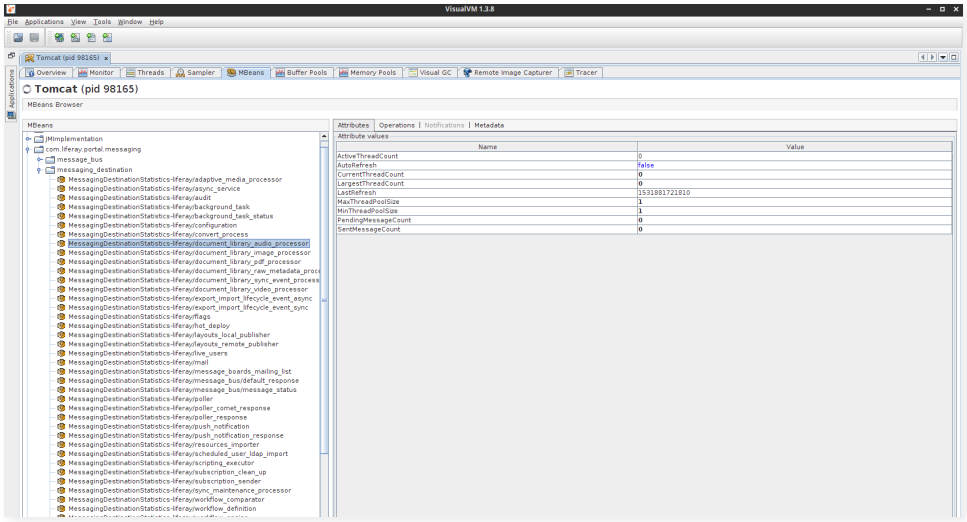

Figure 5 - Monitoring Liferay Message Bus

In the above picture, we see the statistics for the messaging destination "liferay/mail":

- **ActiveThreadCount:** Displays the current number of active worker threads for this destination.
- **CurrentThreadCount:** Displays the total number of worker threads.
- **LargestThreadCount:** Displays the maximum number of worker threads. This is the high water mark reached when the destination was the busiest.

- **MaxThreadPoolSize:** Displays the maximum number of worker threads allowed for this destination. The destination will not allocate more than this number of threads to service requests.
- **MinThreadPoolSize:** Displays the minimum number of worker threads that should be started when the destination is activated.
- **PendingMessageCount:** Displays the number of messages waiting for worker threads to deliver them.
- **SentMessageCount:** Displays the total number of messages delivered via this destination.

Portlet CSS

Liferay DXP provides you the ability to assign custom style sheets within your portlets. You can choose to deactivate this feature, especially if you are not planning on deploying any custom portlets in DXP. You can do so by adding the following to portal-ext.properties:

```
portlet.css.enabled=false
```

Servlet Filters

Liferay DXP ships with a large collection of servlet filters to implement features like compression, SharePoint support, SSO, etc. You can improve performance by disabling those that you are not using.

You can disable the following by modifying portal.properties:

Strip Filter: Used to remove extraneous whitespaces in generated HTML. Web servers should be used to handle this processing. To deactivate, add the following to portal-ext.properties:

```
com.liferay.portal.servlet.filters.strip.StripFilter=false
```[5]

SharePoint Filter: Used to enable DXP to understand SharePoint protocols for integration with MS Office applications. To deactivate, add the following to portal-ext.properties:

```
com.liferay.portal.sharepoint.SharepointFilter=false
```

5 In place of the GZIP and Strip filters, we recommend using the PageSpeed Apache Module (). This module does require deploying the Apache HTTP server in your Liferay architecture.

The following filters have been moved to OSGi modules and are disabled by default:

SSO CAS Filter: Used to implement single sign-on using CAS.

SSO NTLM Filter: Used to implement single sign-on using NTLM.

SSO OpenSSO Filter: Used to implement single sign-on using OpenSSO.

You can check the status of these filters in System Settings.

Session Timeout

With most web applications, the default session timeout is configured for 30 minutes. Although this may be friendly for the user, it does create added resource consumption for the application. Liferay DXP provides several techniques to help you reduce the session timeout for idle users while minimally impacting usability. To reduce the lifespan of the session, you should modify the web.xml[6] and change the timeout:

```
<session-config>
  <session-timeout>10</session-timeout>
</session-config>
```

Template Caching

Liferay utilizes Velocity and Freemarker templating engines as part of its WCM, Theming, ADT and other features. By default, they are configured to frequently check whether template files have been modified. These settings allows for developers to work quickly, but the settings should be modified when going into production.

By default, the settings designate templates to remain in cache for 60ms. For production use, you can increase this for instance to 600000ms (10min), 3600000ms (1 hour) or -1 (indefinite).

To change the values, you may use System Settings to find settings for FreeMarker Engine and Velocity Engine. The property to modify this will be labeled as “Resource Modification Check Interval.”

6 On Tomcat, the DXP web.xml is located in $CATALINA_HOME/webapps/ROOT/WEB-INF/web.xml

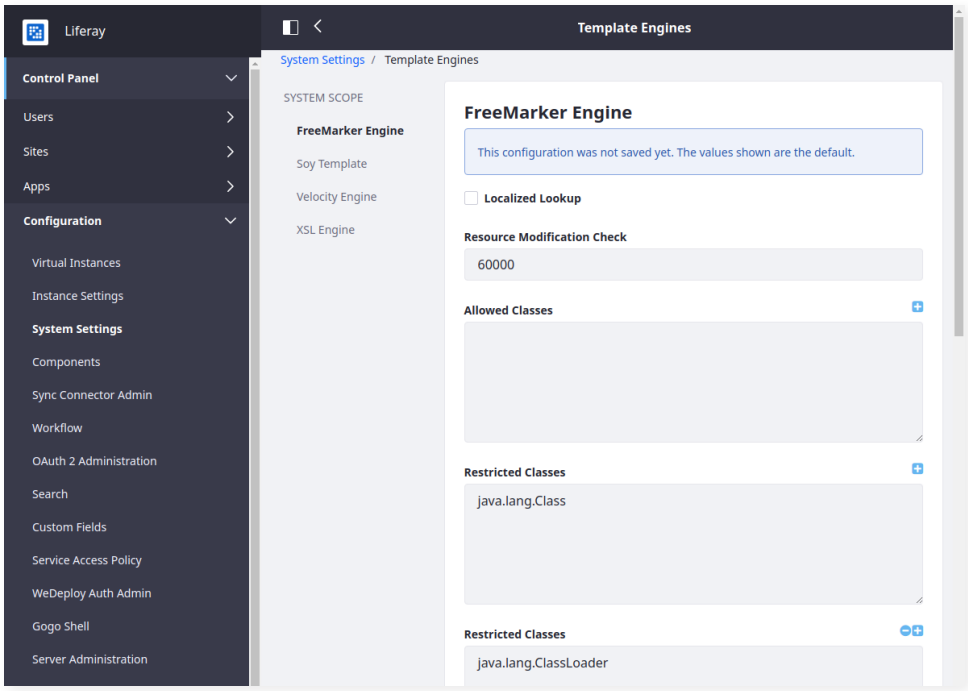

Figure 6 - Freemarker Engine System Settings

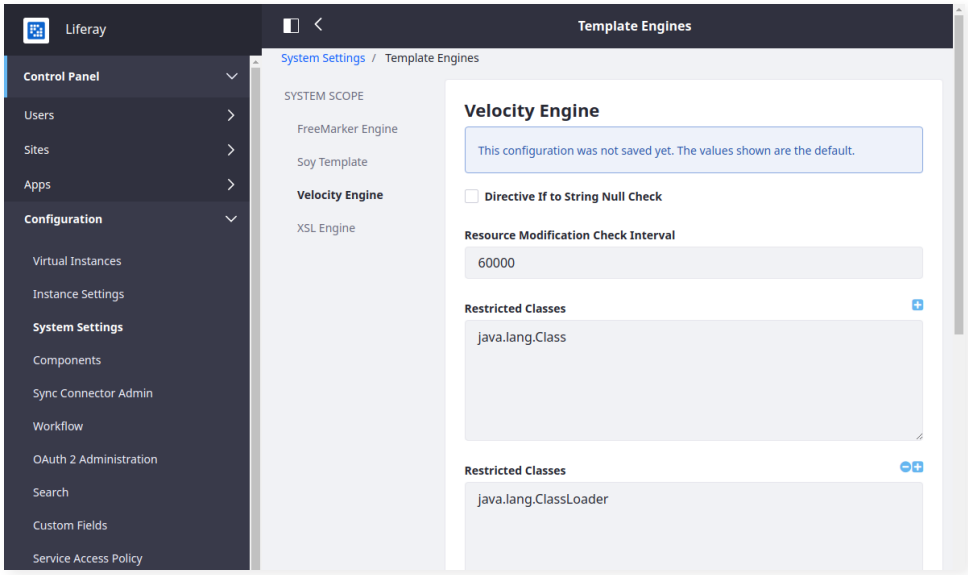

Figure 7 - Velocity Engine System Settings

User Session Tracker

Liferay DXP enables administrators to view the activities of the users currently using the system. While useful for troubleshooting, this feature may decrease performance. In general, we recommend deactivating this feature. You may do so by adding the following to portal-ext.properties:

```
session.tracker.memory.enabled=false
```

Liferay Enterprise Search

In Liferay DXP, Liferay ships with an embedded Elasticsearch search engine (i.e., the Elasticsearch engine runs in the same JVM as Liferay DXP). Although this solution is great for having out-of-the-box search in Liferay, it will not officially be supported by Liferay for production use, only for development. For production usage, Liferay will only support usage of the Elasticsearch search engine running outside of the DXP JVM (i.e., 1 JVM for Liferay DXP and a separate JVM for the Elasticsearch search engine).

Search engines benefit heavily from caching and their JVM memory profiles are substantially different from a JVM focused on serving content and web views (e.g., Liferay JVM). For these reasons, the two applications should always be kept separate in production environments.

The following sections provide a synopsis of Elasticsearch configurations. Prior to deployment, we strongly recommend reading Elastic's documentation on production deployment in "Elasticsearch – the Definitive Guide."

Sizing Your Deployment

When sizing your Elasticsearch deployment, you must carefully consider your CPU, memory, disk and network capacity. As a general rule of thumb, you should strive to deploy Elasticsearch on medium to large machines[7] to scale effectively while avoiding large quantities of machines. Also, you should avoid running multiple Elasticsearch JVMs on the same operating system.

CPU

Liferay recommends at least eight CPU cores allocated to the Elasticsearch engine. This assumes only one Elasticsearch JVM running on the machine.

7 Medium machines generally have at least 2-4 vCPUs. Large machines generally have at least 4-8 vCPUs.

MEMORY

Liferay recommends at least 16GB of memory. The preference is for 64GB of memory.

DISK

Search engines store their indices on disk and thus disk IO capacity can greatly impact search performance. Liferay recommends deploying Elasticsearch on SSD when possible. If you are unable to use SSD, Liferay recommends high performance disks (15k RPM drives). You should also consider using RAID 0 for both SSD and traditional hard disks.

In general, you should avoid using NAS (network attached storage) for Elasticsearch as the network overhead can be quite large. If you are using public cloud infrastructure like Amazon Web Services, this means you should rely upon instance local storage and avoid network storage like Elastic Block Store (EBS).

You must ensure you have at least 25 percent more disk capacity than the total size of your indices. For instance, if you have 50GB of data to be indexed you should plan for having at least 65GB of disk space available. Index sizes will vary based on the indexed content.

CLUSTER SIZE

Even though DXP can work with an Elasticsearch Cluster comprised of one or two nodes, the minimum cluster size recommended by Elastic for fault tolerance is three nodes.

NETWORKING

Elasticsearch relies upon clustering and sharding to deliver fast, accurate search results. Consequently, it relies upon a fast and reliable network. Most modern data centers provide 1GbE or 10GbE between machines. You should avoid spreading Elasticsearch clusters across multiple data centers. Elasticsearch does not support multi-data center deployments, especially data centers spread across large distances (e.g., cross continents).

Configuring Elasticsearch

Prior to starting Elasticsearch, you must configure a few properties in your elasticsearch.yml:

- `cluster.name`
    - By default Liferay DXP expects this to be "LiferayElasticsearchCluster".
    - If you wish to configure with a different cluster name (e.g., "elasticsearch_production"), then you must also modify Liferay DXP's Elasticsearch configuration in System Settings. The cluster name configured in Liferay must match the one configured on the Elasticsearch server.

- `node.name`
  - Each node in your Elasticsearch cluster should be given a unique name.
- `discovery.zen.ping.unicast.hosts`
  - We strongly recommend using unicast for discovery versus multicast. This prevents nodes accidentally joining the cluster.
  - Not every Elasticsearch node in the cluster needs to see each other at startup. If a new node can connect to one member of the cluster at startup, then it will automatically receive the topology information and communicate with other nodes.
- `thread_pool.bulk.queue_size`
  - Liferay uses bulk requests to reduce the number of network calls between Liferay and Elasticsearch. It is advisable to increase the queue size for the bulk request thread pool to at least 100.

In addition to the above settings, you may choose to configure the following for debugging purposes:

```
# The following overhead settings are percentages
monitor.jvm.gc.overhead.debug: 40
monitor.jvm.gc.overhead.info: 70
monitor.jvm.gc.overhead.warn: 90
```

Tuning Your Deployment

JVM

In general, 45 percent of the available system memory should be allocated to Elasticsearch, up to a maximum of 31GB. In general, no other JVM settings should be adjusted within Elasticsearch. You should configure heap sizing by setting the environment variable: ES_HEAP_SIZE.

The JVM vendor and version used for the Elasticsearch server must be identical to the version used for Liferay DXP.

FILE SYSTEM

You should configure your OS for at least 64,000 file descriptors. The default Linux value is 1024.

Elasticsearch also uses NioFS and MMapFS. Consequently, you must ensure there is sufficient virtual memory available for memory-mapped files.

Please consult your system administrator on how to configure these values.

Monitoring Your Deployment

Elasticsearch provides a monitoring tool called X-Pack Monitoring. X-Pack Monitoring is comprised of two components:

- A server side component residing in your Elasticsearch nodes that feeds performance data to another Elasticsearch node.
- A series of Kibana dashboards to help you visualize Elasticsearch performance information.

These dashboards will help you monitor the health and performance of your Elasticsearch cluster in order to better tune and manage your search cluster.

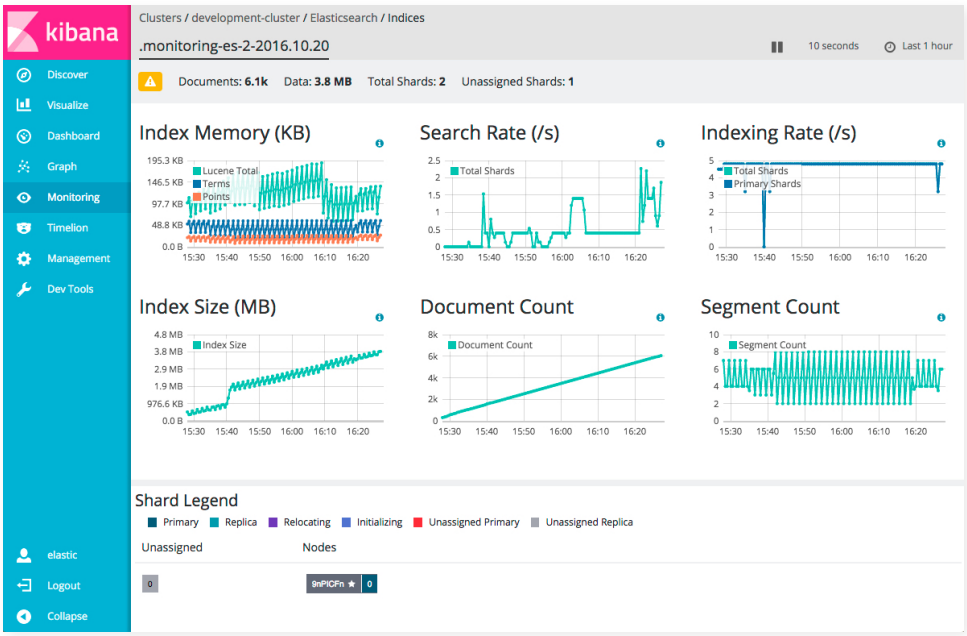

Figure 8 - Sample X-Pack Monitoring Dashboard

To use X-Pack Monitoring with Liferay's Elasticsearch cluster, you must have Liferay's Enterprise Search subscription. For more information, please contact your account executive.

Securing Your Deployment

In order to perform search operations, a search engine must index and store a large quantity of data in its indices. By default, access to this data is not secured. Unlike databases that require login for access, search engines do not require logins to perform searches and browse its stored information.

Elasticsearch's X-Pack Security adds over-the-wire encryption and authenticated search engine access. Over-the-wire encryption ensures all communications between Elasticsearch nodes are encrypted and the communication between Liferay and the Elasticsearch servers are protected. Authenticated search engine access ensures all connections with Elasticsearch require authentication, much like a database.

These security capabilities are only available with a Liferay Enterprise Search subscription. For more information, please contact your account executive.

Liferay Upgrade Tool

Liferay provides a tool in Liferay DXP to execute the upgrade process from older versions. This tool runs a java process which is in charge of executing every modification needed to adapt the database to the new version.

During an upgrade process different elements such as the database, the JVM, the CPU, the network and the disk (IO operations and filesystem) play a very important role in performing a successful upgrade and in reducing the time it takes to upgrade. In the following sections we will analyze how to size them accordingly.

For a comprehensive documentation of this upgrade process, please refer to the 7.2 database upgrade article in the Help Center.

Sizing Your Upgrade

Tuning and sizing all these elements accordingly for your own use case is key to complete the upgrade successfully. Below are some recommendations on how to measure and tune each component.

DATABASE

An upgrade is a process which manipulates data to adapt it to the new version. For this reason an upgrade's environment and database activities differ from those in production:

- Upgrade processes execute many more update statements (INSERT, UPDATE or DELETE) than SELECT statements.
- It's a completely safe environment because it's executed on a database backup on a DXP server separate from the current production server.

Taking these differences into account, it's important to tune the database properly to get the maximum benefit from it. Liferay supports several database vendors and each provides different ways to tune or configure the database server but there are some general recommendations that needs to be applied in order to improve the upgrade performance:

- Deactivate data integrity measures that impact performance. Restore the backup if there is a failure during the process.
- Make commit-related I/O operations during transactions asynchronous.
- Increase the interval to flush commits to disk.
- Adjust the database log file capacity based on the total log capacity used in the test upgrade and the database vendor's recommendation.

Note 1: The previous database recommendations are only valid for DXP upgrades. Never use this configuration to run DXP.

Note 2: On databases that have multiple schemas, global properties and configurations affect all the schemas. Determine whether you should isolate your test schema to its own database before changing global properties or configurations.

JAVA TUNING

Although database's tune is the key to success, java tuning is also important to execute the upgrade under the proper conditions. The following parameters are helpful to complete the upgrade successfully:

```
-Dfile.encoding=UTF-8 -Duser.country=US -Duser.language=en
-Duser.timezone=GMT -Xmx4096m
```

The Xmx parameter is the most important and has to be defined depending on the size of the database. The upgrade process usually gets and manipulates the data in batch mode but the size of certain tables and records can consume most of the memory. Allocate this parameter properly is important in those cases. 4Gb or 6Gb can be a good starting point but if you have a big enough database (above 20Gb) or journalArticle table size is bigger than 10Gb probably it would require 10Gb or above as Xmx parameter.

CPU

Several processes during the upgrade take advantage of multithreading and parallel execution based on the number of CPU Cores. For this reason, having a CPU with at least four cores will reduce the upgrade time.

NETWORK

Due to the high number of operations sent to the database in a short space of time it is mandatory that the net latency between the upgrade tool server and database server is low. Even if you haven't planned to install those servers in the same network, you should do it to execute the upgrade process.

DISK AND I/O OPERATIONS

During the upgrade process several operations are performed on the disk such as:

- Manipulating document library binary files.
- Creating new document library binary files (upgrading from 7.0 or below where journal article images are migrated to the document library).
- Generating backup files for certain database tables.

For this reason there should be enough free disk space to handle these operations and the I/O latency has to be low. The following rule could you give an idea about the maximum free space you need if you upgrade from 7.0 or below:

```
Max free disk space to allocate = size of biggest database
tables + (number of records in JournalArticleImage
table * medium size of image files)
```

If you upgrade from 7.1 the need of space is lower, you can apply the following formula:

```
Max free disk space to allocate = size of biggest database tables
```

Backup Strategy During the Upgrade

There are a few things to consider in the backup strategy:

- Backup your database and document library before an upgrade is always mandatory. Both the database and the document library files need to be synchronized.
- Depending on the size of your database you can consider performing another backup at an intermediate point. Setting an intermediate point between backups is useful for large databases requiring hours to execute an upgrade, because it prevents the need to execute an upgrade again when issues are found with the data during the upgrade process.

Let's see some different scenarios:

CASE 1: UPGRADE FROM 6.2 OR ABOVE

In these cases is recommended to upgrade directly to DXP 7.2 with one single backup first. In case the upgrade takes hours and you find any issue during it, consider the following strategy in the next try:

1. Upgrade only the Core. You can get more information about how to do it here.
2. Make a backup of database and document library at that point.
3. Upgrade the modules.

This strategy makes more sense in non-production environments since in those cases you can fix the data/issue and come back to that intermediate point to repeat the process. In production there is not usually time for these kind of operations so executing the upgrade in shot makes more sense.

CASE 2: UPGRADE FROM 6.1 OR BELOW

In this case, you need to perform the upgrade in two steps since there is no direct path to upgrade from Liferay 6.1 or below to Liferay 7.2. You should follow this approach:

1. Upgrade first to the most recent Liferay version which allows you to go directly from your original Liferay version. Please check the following table as reference:

Original Liferay version	Steps to upgrade your database
6.0.x (except 6.0.12)	Upgrade to 7.0 Upgrade to 7.2
6.0.12 and 6.1.x	Upgrade to 7.1 Upgrade to 7.2

2. Make a backup of database and document library at that point.
3. Upgrade to Liferay DXP 7.2.

Similarly to the Case 1, making a second backup of the database and document library in an intermediate step likely will not make sense in production environments, where there is no time for recovering a backup.

Summary

In the preceding sections, we outlined the steps to design a fully fault-tolerant Liferay Digital Experience Platform deployment. The architecture described builds a solid foundation for future growth. In addition, the Liferay Engineering team has shared several key factors to successfully tune a Liferay DXP deployment. Although these parameters have wide applications and have withstood many different load testing scenarios, Liferay Engineering recommends continuously monitoring your Liferay DXP deployment to ensure long-term performance.

Disclaimer

Liferay can only give you an initial tuning recommendation based on benchmarks that have been performed on the out-of-the-box product. It is up to you as system architects and business analysts to come up with the utilization scenarios that your system will need to service. It is your responsibility to run the appropriate load tests on your system before production deployment, so that you can identify significant bottlenecks due to custom applications/portlets and other unforeseen system and network issues and implement appropriate configurations.

Moving Forward

Liferay DXP Cloud

Liferay DXP can be deployed quickly and managed in the cloud with Liferay DXP Cloud, an enterprise PaaS offering available with a separate subscription. Liferay DXP Cloud takes care of many infrastructure needs to free up IT resources to focus on building the new applications and features that bring value to the business. See Liferay DXP Cloud for more information.

Liferay and Dynatrace

Liferay has a partnership with Dynatrace, an industry leading application performance management (APM) solution. With Dynatrace and the Fastpack for Liferay, customers can gain deeper performance insight into their Liferay DXP deployment. Learn more at liferay.com/dynatrace-apm.

Liferay Global Services

Liferay Global Services has specialists that focus on Liferay DXP Go Live and performance tuning consultation. On-premise and hosted cloud configurations are also supported by Liferay with multiple hosting and managed hosting providers available today including AWS EC2, Azure and more. Learn more at liferay.com/consulting.

Liferay makes software that helps companies create digital experiences on web, mobile and connected devices. Our platform is open source, which makes it more reliable, innovative and secure. We try to leave a positive mark on the world through business and technology. Hundreds of organizations in financial services, healthcare, government, insurance, retail, manufacturing and multiple other industries use Liferay. Visit us at liferay.com.

201116