# Liferay Digital Experience Platform 7.1

デプロイメントチェックリスト

# 目次

# はじめに

ライフレイのエンジニアリングチームとグローバルサービスチームはこれまでに、ライフレイ・デジタルエクスペリエンス・プラットフォーム（以下 Liferay DXP）7.1 のパフォーマンスと拡張性を徹底的にテストしてきており、そのテストを通じて蓄積された知見をこのチェックリストにまとめました。Liferay DXP のデプロイメントのパフォーマンスプロファイルはいくつかの要因によって異なりますが、このチェックリストでは、チューニングを監視する上で必要不可欠なパラメーターと、出発点となる各種初期設定の一覧を掲げます。ここに掲げる初期設定は、ライフレイエンジニアリングの拡張性チームによる厳しいテストを経たものです。

また、豊富な経験と専門的知識をもったライフレイエキスパートによる本番前のチューニングや設定のコンサルティングサービスも提供しています。ご質問がございましたらぜひ一度、こちらからお問い合わせください。

# リファレンスアーキテクチャ

適切なアーキテクチャの選択が、デプロイメント行程で最初に行う意思決定の一つです。適切なアーキテクチャを選択するには以下の点を考慮する必要があります。

- 情報セキュリティ：機密性の高いハードウェアや情報を悪意のある攻撃や侵入から保護すること
- パフォーマンス：目的とする総ユーザー数や、同時トランザクションに対応すること
- 耐障害性：予期せぬ障害または定期保守の際のアップタイムを維持すること
- 柔軟性と拡張性：大幅な再設計を行うことなく追加機能や追加ユーザーに対応できるように拡張可能なアーキテクチャを設計すること

少し複雑に見えますが、図 1 に示すリファレンスアーキテクチャには高レベルの耐障害性と柔軟性が備わっています。


図 1 – Liferay DXP リファレンスアーキテクチャ

このアーキテクチャは以下の階層で構成されています。

- ファイアウォール : 侵入検知と防御
- ロードバランサー層 : 複数のウェブサーバーのリソース間での円滑な負荷分散
- ウェブサーバー層 : 画像、リッチメディア、CSS ファイルなどの静的コンテンツ要素を配信します。CA Siteminder、Oracle Identify、Ping などのようなシングルサインオン・ソリューションの統合モジュールも提供します。コンテンツ配信ネットワーク(CDN) は、ウェブサーバーが行っている静的ファイルキャッシュやエッジキャッシュの一部を行う場合があります。しかし、URL 書き換えのような機能をより簡単に実行するにはこの階層が依然必要な場合があります。
- アプリケーション層 : Tomcat、JBoss、Oracle Weblogic、IBM Websphere などのライフレイが対応しているアプリケーションサーバーをホストします(この他のアプリケーションサーバーについては、Liferay DXP 7.1 互換性マトリクスをご覧ください)。Solr や Elasticsearch などの検索エンジンもホストします。

- データベース層 : MySQL、Oracle、MS SQL、IBM DB2、Postgres などのライフレイが対応しているデータベースサーバーをホストします（この他のプラットフォームについては、Liferay DXP 7.1 互換性マトリクスをご覧ください）

各階層内でデプロイされるハードウェアはトランザクションの種類によって異なります。ライフレイは、ハードウェア仕様ガイドとしてライフレイエンジニアリングの以下のベンチマーク環境を使用します。

**ロードバランサー層**

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

**ウェブ層**

Apache、Nginx、Varnish 等を使用したキャッシュ圧縮およびその他の機能を備えています。

- CPU: Intel Core 2 Duo E6405 2.13GHz x1、メモリ : 4GB、ストレージ : 146GB 10,000rpm SCSI

**アプリケーション層**

このアーキテクチャの中心になります。

- CPU: Intel Xeon E5-2643 v4 3.40GHz x2、メモリ : 64GB、ストレージ : 300GB 15,000rpm 6Gbps SATA x2 – ライフレイポータルに使用
- CPU: Intel Xeon E5-2643 v4 3.40GHz x2、メモリ : 64GB、ストレージ : 300GB 15,000rpm 6Gbps SATA x2 – Elasticsearch に使用

**データベース層**

- CPU: Intel Xeon E5-2643 v4 3.40GHz x2、メモリ : 64GB、ストレージ : 300GB 15,000rpm 6Gbps

アプリケーションサーバーには 64GB の物理メモリが搭載されていますが、Java 物理マシン（JVM）が使用するヒープサイズが大きくない場合は、これよりも小さいメモリを選択しても構いません。現在のオペレーティングシステムは使用可能な物理メモリをファイルシステムキャッシュにも使用します。

## 仮想化デプロイメントとクラウドデプロイメント

リファレンスアーキテクチャは物理的デプロイメントを示していますが、同じ概念をクラウドベースまたは仮想化されたデプロイメントに当てはめることができます。ライフレイのお客様の多くは、パブリッククラウド上（例えば、Amazon EC2）か、自社のプライベートクラウド上（例えば、VMWare VSX ベースのプライベートクラウド）へのデプロイを選択しています。各物理マシンは適切な数の仮想マシンに置き換えることができます。

仮想化デプロイメントでは、十分な CPU リソースを割り当てることが非常に重要です。例えば、Amazon AWS にデプロイするシステムでは、割り当てる CPU は Amazon EC2 計算ユニットを使って計算されます。しかし、1 計算ユニットは 1 物理 CPU と同等ではなく、CPU 上の 1 コアとも同等ではありません。Amazon の条件に従えば、リファレンスアーキテクチャで使用される各アプリケーションサーバーは「Cluster Compute Quadruple Extra Large Instance」または 33.5 EC2 計算ユニットに相当します。したがって、仮想化／クラウドデプロイメントを適切に計画するには、仮想化のオーバーヘッドだけでなく十分な CPU リソースを割り当てることも考慮する必要があります。

## 耐障害性

リファレンスアーキテクチャはあらゆるレベルで耐障害性に対応しています。ウェブ層、アプリケーション層、データベース層がクラスタ化されているため、いずれかのノードで重大なハードウェア障害が発生しても、パフォーマンスをほとんど低下させることなくユーザーにサービスを提供し続けることができます。

ここに示しているリファレンスアーキテクチャは、単一のデータセンター内での適切な耐障害性を確保するための最低限のデプロイメントユニットです。負荷パターンに応じて各階層内のサーバー数を増やすことで、十分な耐障害性を維持しつつ、そのデプロイメントがサポート可能なユーザー数を乗数的に増やすことができます。

複数のデータセンターを使用して耐障害性を実現するアーキテクチャは、このリファレンスアーキテクチャには含まれていません。

## パフォーマンス

各デプロイメントのパフォーマンス特性は、アクティビティのタイプやカスタムアプリケーション要素のパフォーマンスによって異なります。ライフレイエンジニアリングでは、エンタープライズ仕様のコンテンツ管理、コラボレーション、ソーシャルネットワークに関して、Liferay

DXP の（追加設定しない状態での）パフォーマンスをベンチマーク評価するための一連のシナリオを作成しています。これらのリファレンスアーキテクチャの結果は、Liferay DXP が 46,750 人以上の仮想コラボレーションユーザーと、平均ログイン時間 250 ミリ秒で 1 分当たり 79,000 回以上のログインに対応できることを示しています。これだけの負荷をこのリファレンスアーキテクチャ内で実現した一方で、CPU リソースの使用率はウェブ層で 40% 以内、アプリケーション層で 95%、データベース層では 5% 未満にとどまりました。

### 拡張性

ライフレイエンジニアリングによるテストを通じて、Liferay DXP は直線的にスケーリングできることが分かっています。したがって、1 台のアプリケーションサーバーが X 人の仮想ユーザー数をサポートしている場合、十分なリソースがデータベースサーバーとウェブサーバーにあると仮定すると、必要なアプリケーションサーバーの合計数を計算できます。

### セキュリティ

ロードバランサー層の前に配置するファイアウォールは、侵入検知・防御に十分対応していますが、組織の情報セキュリティ要件に応じて各階層間にファイアウォールレイヤを追加し、さらにインフラを保護することができます。

## Liferay DXP チューニングガイドライン

DXP のチューニングを行う際には考慮すべき点がいくつかあります。Liferay DXP に固有のものと、すべての Java と Java エンタープライズアプリケーションに当てはまるものがあります。以下のガイドラインは、デプロイメントのチューニングを行う際の最初のベースラインとして活用していただくことを意図しています。.

### アプリケーションサーバーのチューニング

実際の設定名称は異なる場合がありますが、各概念はほとんどのアプリケーションサーバーに当てはまります。ここでは、Tomcat を例にとって、アプリケーションサーバーチューニングの概念について説明します。個別の詳細な設定については、アプリケーションサーバープロバイダーのドキュメンテーションも参考にしてください（アドバイスが記載されているはずです）。

## データベース・コネクションプール

データベース・コネクションプールのサイズは、通常、スレッドプールサイズの約 30 ～ 40% です。コネクションプールは、DXP がデータベースからデータを取得する必要がある場合(ユーザーログイン時など）にデータベースに接続します。コネクションプールのサイズが小さすぎると、接続要求がサーバーのキューに格納され、データベースへの接続が待たされることになります。一方、コネクションプールのサイズが大きすぎると、データベースの接続に使用されないプールが発生し、リソースが無駄に消費されることになります。

スレッドプールについても同様に、パフォーマンステストに基づいてこれらの設定をモニターし、調整する必要があります。

**Tomcat 9.0.x**

Tomcat では、コネクションプールは $CATALINA_HOME/conf/ Catalina/localhost/ROOT.xml の Resource 要素に設定されます。

```
<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&am
p;dontTrackOpenResources=true&amp;holdResultsOpenOverStatementC
lose=true&amp;useFastDateParsing=false&amp;useUnicode=true”
/>
```

この設定では、10 個の接続で開始し、最大 85 個の接続がプールされます。

C3P0、HikariCP、Tomcat など、さまざまなデータベース接続プールプロバイダーから選択できます。portal.properties にはライフレイの JDBC 環境を設定することも選択できます。JNDI あるいは portal.properties の選択は、使用するアプリケーションサーバーと自社の IT 規格に基づいて行うとよいでしょう。例えば、Tomcat を使用しており、社内規格が JNDI を JEE のリソースとして宣言することを指示していない場合は、portal-ext.properties を使用することがおそらく最も簡単な方法です。

## JSP エンジン内の開発環境を無効にする

ほとんどのアプリケーションサーバーの JSP エンジンは、開発モードに設定されています。本番稼働を開始する前にこれらの設定の多くを無効にすることをライフレイでは推奨します。

- 開発モード : JSP コンテナが JSP ファイルの変更についてファイルシステムにポーリングできるようになります。
- マップトファイル : JSP テキスト 1 行当たり 1 つのステートメントに対して、1 つのプリントステートメントを持つ静的コンテンツを生成します。

**Tomcat 9.0.x**

$CATALINA_HOME/conf/web.xml で、JSP サーブレットを以下のように更新します。

```
<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>
```

## スレッドプール

アプリケーションサーバーへの各着信要求によって、その要求時間分のワーカースレッドが消費されます。要求を処理する使用可能なスレッドがない場合、その要求はキューに格納され、ワーカースレッドが使用可能となるまで待たされます。細かく調整されたシステムでは、スレッドプール内のスレッドの数は、同時要求の合計数と比較的バランスが取れているはずです。サービスの要求を待っているアイドル状態のスレッドが大量に発生しないようにする必要があります。

ライフレイエンジニアリングでは、最初に 50 個のスレッドを設定し、アプリケーションサーバー内のモニタリングコンソールでモニターすることを推奨しています。平均ページ実行時間が 2 〜 3 秒の範囲内であれば、より多くの数（例えば、250 個）のスレッドを使いたいと思うかもしれません。スレッドプール内のスレッドが少なすぎると大量の要求がキューに格納され、プール内のスレッドが多すぎるとコンテキストスイッチが多く発生することになります。

**Tomcat 9.0.x**

Tomcat では、スレッドプールは $CATALINA_HOME/conf/server.xml の中の Connector 要素に設定されています。詳細は、Apache Tomcat documentations で確認できます。ライフレイエンジニアリングでは、以下の設定を使用してテストを実施しました。

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

Connector 要素では、コネクタのタイプや接続タイムアウト、TCP のキューといったチューニングパラメーターを利用できます。詳細は、該当する Tomcat のドキュメンテーションを参照してください。

## Java 仮想マシンのチューニング

JVM のチューニングはガーベジコレクタと Java メモリヒープのチューニングが中心となっています。これらのパラメーターは、実行するアプリケーションのスループットを最適化することを目的としています。ここでは Oracle の 1.8 JVM をリファレンスアーキテクチャに使用しています。対応しているその他の JVM のバージョンや実装を選択することもできます。互換性のあるその他の JVM については Liferay DXP7.1 互換性マトリクスを参照してください。

### ガーベジコレクタ

適切なガーベジコレクタ（GC）を選択することによって Liferay DXP デプロイメントの即応性を向上させることができます。ライフレイでは、停止時間の短いコンカレントコレクタの使用を推奨しています。

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

パラレル・スループットコレクタや G1 コレクタなどの他の利用可能な GC アルゴリズムから選択できます。ライフレイでは、最初に、新世代のパラレルコレクタと旧世代のコンカレント・マークスイープ・コレクタ（CMS）を使ったチューニングから始めることを推奨します。

**注記 :** “ParallelGCThreads=16” の 16 の値は、利用可能な CPU の種類によって変わります。CPU の仕様に従ってこの値を設定することを推奨します。

Linux マシンでは、“cat /proc/cpuinfo” を実行することによって CPU 数が分かる場合があります "

**注記 :** G1 などの「新しい」アルゴリズムがありますが、ライフレイエンジニアリングが行った G1 のテストでは、パフォーマンスは向上しませんでした。お使いのアプリケーションによってはパフォーマンスが異なる場合があるため、G1 をテストとチューニングの計画に加えるとよいでしょう。

## Java ヒープ

Java のメモリヒープのチューニングを検討する際、ほとんどの方は、最大ヒープメモリと最小ヒープメモリの設定を検討されるはずです。しかし、多くのデプロイメントにおいて、最適なパフォーマンスを実現するためには、はるかに高度なヒープチューニング（若い世代の領域のサイズ、旧世代存続期間、サバイバー領域、その他多くの JVM 内部の構造のチューニングなど）が必要です。

ほとんどのシステムについて、ライフレイでは、最低でも以下のメモリ設定から始めることを推奨します。

```
-server -XX:NewSize=700m -XX:MaxNewSize=700m -Xms2048m -Xmx2048m
-XX:MetaspaceSize=512m -XX:MaxMetaspaceSize=512m -XX:SurvivorRatio=6
-XX:TargetSurvivorRatio=90 -XX:MaxTenuringThreshold=15
```

大きなヒープサイズ（例えば、4GB 超）を必要とするシステムでは、大きいページサイズを使用すると効果的な場合があります。以下の JVM のオプションを使用して大きいページサイズを有効にすることができます。

```
-XX:+UseLargePages -XX:LargePageSizeInBytes=2m
```

アプリケーションのプロファイルに基づいて異なるページサイズの指定を選択することが可能です。

**注記 :** JVM でサイズの大きいページを使用するには、JVM が実行されているオペレーティングシステムを設定してサイズの大きいページを有効にする必要があります。Linux では、“cat /proc/meminfo” を実行し、“huge page” の項目を確認してください。

**注意 :** JVM のヒープに 32GB を超えるメモリを割り当てないようにしてください。

ヒープサイズは、利用可能な CPU リソースの速度と量に応じて設定してください。

## JVM の高度なオプション

ライフレイのベンチマーク環境では、以下の高度な JVM オプションも適用されました。

```
-XX:+UseLargePages -XX:LargePageSizeInBytes=2m -XX:+UseCompressedOops
-XX:+DisableExplicitGC
```

高度な JVM のオプションの詳細については、JVM のドキュメンテーションを参照してください。

上記のパラメーターを合わせると以下のような設定になります。

```
-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=2m -XX:+UseParNewGC -XX:ParallelGCThreads=16
-XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled
-XX:+CMSCompactWhenClearAllSoftRefs -XX:CMSInitiatingOccupancyFraction=85
-XX:+CMSScavengeBeforeRemark -XX:+UseLargePages
-XX:LargePageSizeInBytes=256m -XX:+UseCompressedOops -XX:+DisableExplicitGC
```

**注意 :** 上記の JVM の設定は、パフォーマンスチューニングの開始点であると考えてください。各システムの最終的なパラメーターは、現在のユーザー数やトランザクション速度などさまざまな要因によって変化します。

ライフレイでは、メタ領域とサバイバー領域に十分なメモリが割り当てられていることを確認にするため、ガーベジコレクタの統計情報をモニターすることを推奨します。ガイドラインとして示した上記の数字をそのまま使用すると、メモリ不足障害などが生じる危険性があります。サバイバー領域の不適切なチューニングもヒープ領域の浪費につながります。

## GC と JVM の監視

これまでにご紹介したパラメーターで、JVM のチューニングを問題なく行えるはずですが、GC のパフォーマンスをモニターして、ニーズに適した設定になっていることを確認してください。以下のツールが Oracle JVM のパフォーマンスのモニターに役立ちます。

### Visual VM

このツールには Oracle JVM のパフォーマンス情報（ガーベジコレクタのアクティビティを含む）を表示する集中コンソールが用意されています。

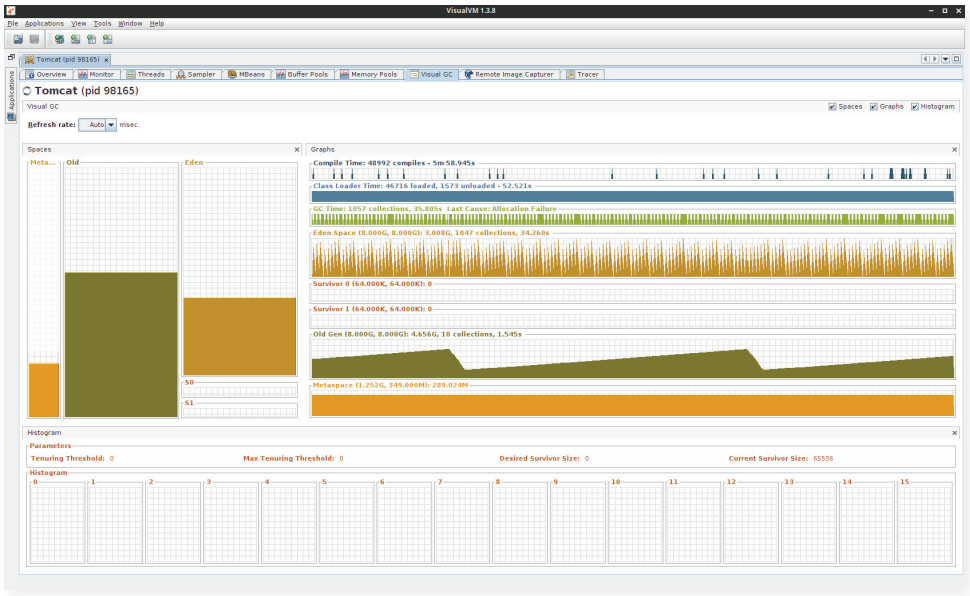


図 2 – Visual VM の Visual GC

### JMX コンソール

このツールは、ライフレイの分散キャッシュのパフォーマンス、アプリケーションサーバースレッドのパフォーマンス、JDBC 接続プールの使用率など[1]、各種統計情報の表示に役立ちます。

アプリケーションサーバーの JVM の引数に以下を追加し JMX 接続を有効にします。

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

1 JMXコンソールは、Tomcatのパフォーマンス情報を観察する際の推奨ツールです。

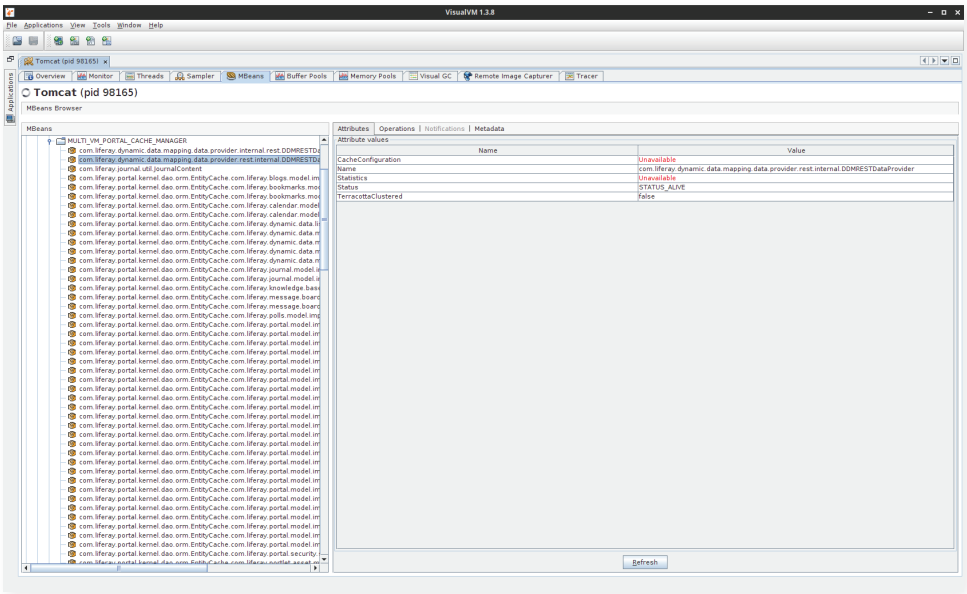


図 3 – Visual VM の JMX コンソール

**ガーベジコレクタの冗長ロギング**

JVM の引数に以下を追加し、JVM ガーベジコレクタの冗長ロギングを有効にします。

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

これらのログは JVM の適切なチューニングに必要となります。

**注記 :** JVM がメモリ不足になった場合に十分なデバッグ情報が得られるように、以下を追加すること検討してください。

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

## Liferay DXP チューニングパラメーター

Liferay DXP は、最適なパフォーマンスを得られるように、各種パラメーターが最適化された状態で出荷されます。しかし、ユースケースによっては別途設定のチューニングを行いたい場合があるでしょう。特に断りのない限り、これらのパラメーターは portal.properties[2] で、あるいはシステム設定を使ってチューニングすることができます。

2 portal.propertiesに格納されているプロパティをオーバーライドする方法、システム設定を使って設定を変更する方法、システム設定を使って設定をエクスポート／インポートする方法については、Liferay DXPの公式ドキュメンテーションを参照してください。.

## キャッシュ

> **警告 :** キャッシュのチューニングには注意が必要です。
>
> キャッシュのサイズはアプリケーションサーバーで使用するメモリ容量に直接影響します。キャッシュのサイズが大きいほど要求処理に利用できるメモリ容量は少なくなります。
>
> 既製のインプロセスキャッシュを拡張しても、使用するアプリケーションのキャッシュ要件を満たせない場合、クラスタ化された、アウトオブプロセスキャッシュとして、Terracotta の使用を検討されることを推奨します。

Liferay DXP は、データベースのやりとりと過剰なオブジェクトの作成を最小限に抑えるため、コンテンツとオブジェクトの両方のキャッシュに依存しています。Liferay DXP は、出荷時の状態では Ehcache を使用してキャッシュのニーズに対応していますが、Terracotta や Oracle Coherence Cache などのキャッシュを追加で設定することが可能です。ここでは、Ehcache を中心に取り上げます。

キャッシュをモニターするには、JMX コンソールを使用する必要があります.[3] 前掲の図に、ユーザー情報の格納に使用されるキャッシュをモニターするための JMX コンソールが表示されています。

**CacheHits** – データベースではなくキャッシュからオブジェクトを取得できた要求の数を表示します。この値には、メモリとディスクからオブジェクトが取得された要求の数も含まれます。

**CacheMisses** – キャッシュにオブジェクトが見つからずデータベースからオブジェクトを取得しなければならなかった要求の数を表示します。

**InMemoryHits** – インメモリ・キャッシュからオブジェクトを取得できた要求の数を表示します。この値には、ディスクストレージからオブジェクトが取得された要求の数は含まれません。

3 キャッシュ使用率の監視には、ライフレイのConnected Servicesも利用できます。

**ObjectCount** – キャッシュの中のオブジェクトの総数

**OnDiskHits** – メモリ内にはオブジェクトが見つからなかったが、ローカルファイルシステム上にオブジェクトが見つかった要求の総数。これが該当するのは、キャッシュのディスクへのオーバーフローが有効化されている場合のみです[4]。

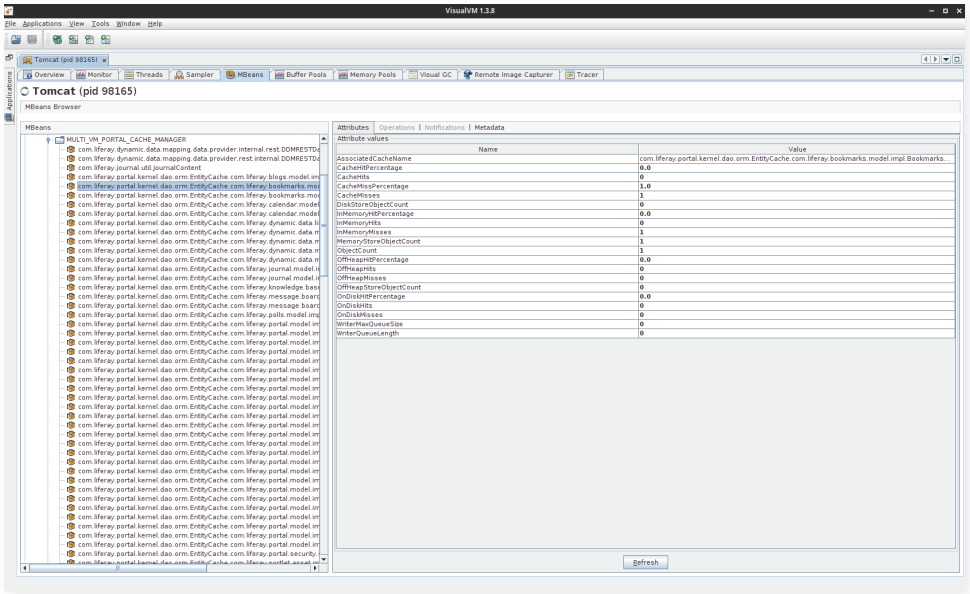


図 4 – ライフレイデータキャッシュの監視

デフォルトでは、Liferay DXP のキャッシュの設定はメモリストアを使用し、最大 10,000 エレメントを格納できます。ディスクストアを使用することでキャッシュのサイズが増える場合がありますが、ディスクの IO 動作への依存が高まるため、この方法は推奨されません。

チューニング例として、ユーザーキャッシュを使用する場合を考えてみましょう。キャッシュの監視を行ったところ、キャッシュミスが多くチューニングが必要であると判断したとします。

この場合、ライフレイのキャッシュ設定を、プラグインで、あるいは portal-ext.properties を変更して、チューニングしオーバーライドすることができます。

Liferay DXP の基本データキャッシュについては、以下のキャッシュがチューニングの対象となります。

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.

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

製品で利用している機能によっては以下のキャッシュが対象になります。

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

## キャッシュレプリケーション

Liferay DXP は、キャッシュされたオブジェクトやイベントをレプリケーションするための高度なアルゴリズム（より効率的なスレッドプーリングを使用するアルゴリズム）が実装された状態で出荷されます。このアルゴリズムが使えるように DXP を設定するには、以下の portal.properties を使ってクラスタリングを有効にします。

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

## カウンターインクリメント

Liferay DXP は、オブジェクト ID の生成に内部のシーケンスジェネレータを使用しています。このカウンターのインクリメントサイズを大きくすると、ID を用意するためにカウンターがデータベースと通信する必要のある回数が減ります。使用するシステムにもよりますが、この数を約 2000 に設定することを推奨します。これは portal-ext.properties に以下を追加することで行えます。

```
counter.increment=2000
```

## ドキュメントライブラリのプレビュー

Liferay DXP のドキュメントライブラリ・プレビュー機能によって、ユーザーはレポジトリに格納されているアセットを、ファイルをダウンロードせずに閲覧できます。このドキュメントプレビュー機能は PDFBox と OpenOffice サーバーの組み合わせ、または ImageMagick を使用します。

PDFBox と OpenOffice サーバーの組み合わせによる方法は、OpenOffice サーバーを使用して最初にドキュメントを PDF に変換し、次に PDFBox を使ってその PDF から画像を生成します。この方法は精度が低く、特定の種類のファイルは適切に表示できない場合があります。この方法を設定するには以下のプロパティを portal-ext.properties に設定してください。

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

また、OpenOffice を「サーバー」モードまたは「ヘッドレス」モードでインストールする必要があります。

ImageMagick を使用する場合は、Liferay DXP がホスティングされている OS に ImageMagick をインストールする必要があります。ImageMagick は imagemagick.org から取得できます。DXP を設定して ImageMagick を使用できるようにするには、以下を設定する必要があります。

```
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
```

ドキュメントサイズや他のパラメーターによっては、ImageMagick をさらにチューニングしたい場合があるでしょう。さらに詳しくは imagemagick.org/script/architecture.php をご覧ください。

ただし、プレビューの生成には費用が相当かかる可能性があります。そのため、デフォルトでは、Liferay DXP はプレビュー生成を以下の別の JVM で実行します。

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

これによって DXP の Java 仮想マシンの安定性を向上させることができます。

## ドキュメントライブラリのストレージ

Liferay DXP のドキュメントライブラリは、Amazon S3 やデータベース、ファイルシステムなどさまざまなストレージエンジンで構成できます。

ライフレイでは、ファイルシステムベースの 2 種類のストレージ（FileSystemStore と AdvancedFileSystemStore）が用意されています。FileSystemStore は、以下のパス構造を使用してファイルを格納します。

*${liferay.home}/document_library/<companyId>/<groupId>/<fileName>/<version>*

AdvancedFileSystemStore は、以下のパス構造を使用してファイルを格納します。

*${liferay.home}/document_library/<companyId>/<groupId>/<fileName_with_extension>/<fileName_without_extension>_<version>/<version>* ディレクトリ階層を追加することによってパフォーマンスと拡張性を向上させることができます。したがい、本番環境では AdvancedFileSystemStore を使用することを推奨します。

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

ライフレイは、ファイルシステムのバックアップ、レプリケーションおよびファイルのロックを行っていないことに留意してください。Liferay DXP では、ユーザーが選択した OS、バックアップツール、ファイルシステムレプリケーションツールを使用してこれらの機能を実現しています。

ファイルシステムのレプリケーションまたはバックアップ機能がない場合は、DBStore を選択するのがよいかもしれません。DBStore はアセットのバイナリデータをリレーショナルデータベースに格納します。これによってデータベースのレプリケーションとバックアップ機能を使ってドキュメントアセットもバックアップできるようになります。ただし、ダウンロードの速度が遅くなります。この方法は、大量のドキュメントのダウンロードが伴う Liferay DXP のデプロイメントでは推奨されません。

ライフレイはストレージ用の JCR アダプターと CMIS アダプターも提供していますが、本番環境での使用は推奨されません。

## ダイレクト・サーブレット・コンテキストのリロード

本番システムでは、サーブレットと JSP コンテナによって JSP をリロードされないようにすべきです。したがって以下の値を portal-ext.properties に設定してください。

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

## 有効化されたロケール

Liferay DXP は多くの言語に対応していますが、ソリューションによっては、利用可能な言語数を減らしたい場合があるかもしれません。完全にサポートされているロケールは以下のプロパティで確認できます。

```
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
```

以下の翻訳が完成していない言語については、完全にはサポートされていません。

```
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
```

これらのロケールは、portal.ext.properties の locales.enabled に追加するだけで使用できます。コンテンツマネージャーの使い勝手を高めるため、サポートすることが必要なものだけに locales.enabled を減らすとよいでしょう。ロケールは、locales.enabled プロパティに追加するだけでいつでも増やすことができます。

## 暗号化アルゴリズム

ライフレイのログイン機能を使用してユーザーパスワードをデータベースに格納しようとしたときに、パスワードに使用する暗号化アルゴリズムをチューニングしたい場合があるかもしれません。デフォルトでは、値は以下のようになっています。

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

これは、128000 回の暗号ラウンドの後 160 ビットのハッシュを生成する「パスワードベースの鍵生成機能」のアルゴリズムです（2014 年の OWASP 勧告）。

しかし、性能の劣る CPU（例えば、仮想 CPU や古い CPU）を使用するデプロイメントでは、他のアルゴリズムを使用したり、ラウンド回数を減らしたりすることが必要となる場合があります。例えば、暗号化ラウンドの回数を減らした場合、CPU の使用率が減るためパフォーマンスが向上します。パスワードのハッシュ値計算に関しては、自社の情報セキュリティ部門にご相談ください。

## 複合 SQL によるグループのクエリ

Liferay DXP は、サイトのような機能を必要とする特定のエンティティを追跡するのに役立つ「グループ」オブジェクトを使用しています。例えば、サイトを持つ組織や、プライベートとパブリックのプロファイルページを持つユーザー等です。組織やユーザーが増えるとグループの数が多くなり複雑になる可能性があります。ライフレイは、これらのグループのクエリを最適化する 2 つの方法を実装しています。

1. 複合 SQL を使用する方法。この方法は、データベース側の使用率が増えます。
2. より多くの処理をメモリ上で実行する方法。この方法は、DXP JVM 上のメモリの消費量と CPU の使用量が増えます。

1 の方法は、特定の種類のグループ（例えば、ユーザー）が多い場合に適しています。2 は、データベースに関する CPU の使用量が減るものの DXP JVM 上の CPU とメモリの使用量が増えます。したがい、いずれにするかはトレードオフを考慮して決めることになります。

複合 SQL のための追加のタイプを設定するには、portal-ex.properties の groups.complex.sql.class.names を更新します。デフォルトでは、com.liferay.portal.kernel.model.User しか含まれていません。以下のような他のモデルを含めたい場合があるかもしれません。

1. com.liferay.portal.kernel.model.Organization: : 組織数が多い場合
2. com.liferay.portal.kernel.model.UserGroup: ユーザーグループ数が多い場合
3. com.liferay.portal.kernel.model.LayoutPrototype and com.liferay.portal.kernel.model.LayoutSetPrototype: レイアウトのバージョンまたはブランチが複数存在するステージング環境を使用する場合

例として :

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

## メッセージバス

> **警告 :** これは高度な設定であるため、修正は注意して行ってください。ライフレイメッセージバスは、ほとんどの状況に対処するようチューニングされています。

Liferay DXP は、メールやインデックス作成などのサービスの多くに、ライフレイメッセージバスを使用した、非同期メッセージングを利用しています。このバスを使用すると、緩く結合されたプラガブルアーキテクチャを実現でき、システムタスク（E メール通知など）をバックグラウンドで実行させることができます。これにより、ユーザーエクスペリエンスが向上します。

ユースケースに応じてメッセージバスのモニターとチューニングを行うことが推奨されます。前述の JMX コンソールでメッセージバスの統計情報を監視することができます。

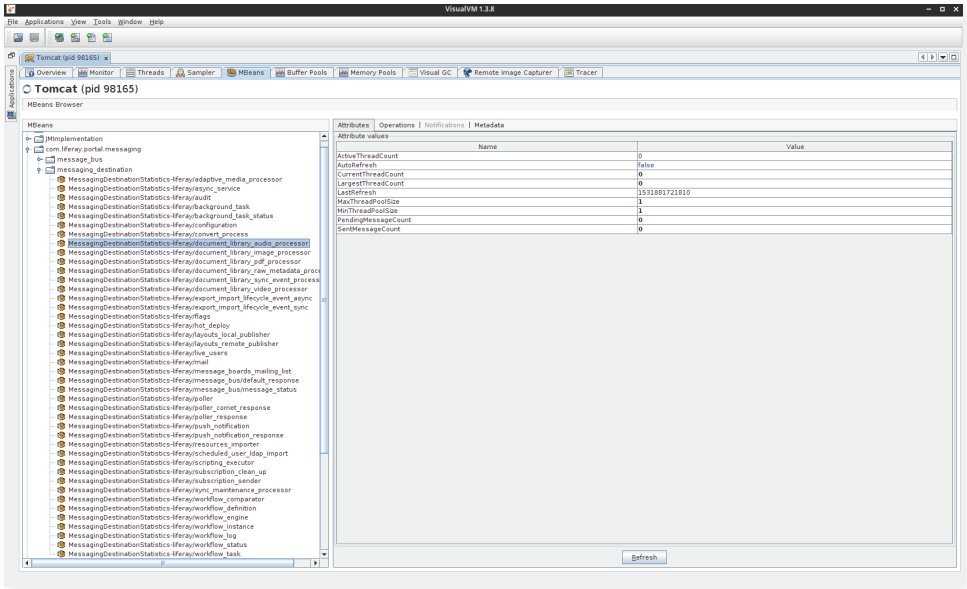


図 5 – ライフレイメッセージバスの監視

上の図では、メッセージング送信先“liferay/mail”の統計情報が表示されています。

- **ActiveThreadCount（アクティブスレッドカウント）**: この宛先に関するアクティブ・ワーカースレッドの現在の数を表示します。
- **CurrentThreadCount（現在のスレッドカウント）**: ワーカースレッドの総数を表示します。
- **LargestThreadCount（最大スレッドカウント）：**ワーカースレッドの最大数を表示します。これは宛先が最もビジー状態に達したときの最大レベルです。
- **MaxThreadPoolSize（最大スレッドプールサイズ）：**宛先が認めるワーカースレッドの最大数を表示します。この宛先はこれより多い数のスレッドはサービス要求に割り当てません。
- **MinThreadPoolSize（最小スレッドプールサイズ）：**宛先が有効化された際に起動する必要のあるワーカースレッドの最小数を表示します。
- **PendingMessageCount（保留中のメッセージ数）：**ワーカースレッドによって配信されるのを待っているメッセージの数を表示します。
- **SentMessageCount（送信済メッセージカウント）：**この宛先によって配信されたメッセージの総数を表示します。

## ポートレット CSS

Liferay DXP では、ポートレット内でカスタムのスタイルシートを割り当てることができます。DXP でカスタムのポートレットをデプロイする計画がない場合などは、この機能を無効にすることを選択できます。それには、portal-ext.properties に以下を追加します。

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

## サーブレットフィルター

Liferay DXP は、圧縮機能や SharePoint 対応機能、SSO 機能といった機能を実現できるように、豊富な種類のサーブレットフィルターが組み込まれた状態で出荷されます。使用しないサーブレットフィルターを無効にすることでパフォーマンスを向上させることができます。

portal.properties を修正することで以下を無効にできます。

ストリップフィルター : 生成された HTML 中の無関係なホワイトスペースを削除するのに使用。この処理にはウェブサーバーを使用する必要があります。無効にするには、portal-ext.properties に com.liferay.portal.servlet.filters.strip.StripFilter=false[5] を追加します。

```
com.liferay.portal.servlet.filters.strip.StripFilter=false
```

SharePoint フィルター : DXP が SharePoint プロトコルを理解できるようにし、DXP と MS Office アプリケーションを統合できるようにするためのもの。無効にするには portal-ext.properties に com.liferay.portal.sharepoint.SharepointFilter=false を追加します。

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

以下のフィルターは OSGi モジュールに移されており、デフォルトでは無効になっています。

SSO CAS フィルター : CAS を使用してシングルサインオンを実装するのに使用。

SSO NTLM フィルター : NTLM を使用してシングルサインオンを実装するのに使用。

5 GZIPとStripフィルターに代えて、Apache用PageSpeedモジュール（mod_pagespeed）の使用を推奨します。このモジュールは、ライフレイアーキテクチャ内にApache HTTPサーバーをデプロイする必要がありません。

SSO OpenSSO フィルター : OpenSSO を使用してシングルサインオンを実装するのに使用。

これらのフィルターの状況はシステム設定で確認できます。

## セッションタイムアウト

ほとんどのウェブアプリケーションは、デフォルトのセッションタイムアウトが 30 分に設定されています。これはユーザーには便利な反面、アプリケーションがより多くのリソースを消費することにつながります。Liferay DXP では、ユーザビリティへの影響を最小限にしつつ、アイドル状態のユーザーのセッションタイムアウトを減らすのに役立ついくつかの方法を用意しています。セッションの持続時間を短縮するには、web.xml[6] を修正し、タイムアウトを以下のように変更してください。

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

## テンプレートキャッシュ

ライフレイは、WCM（ウェブコンテンツ管理）やテーマ作成、ADT（アプリケーション・ディスプレイ・テンプレート）およびその他の機能の一環として、Velocity と Freemarker のテンプレートエンジンを利用しています。デフォルトでは、これらはテンプレートファイルの変更の有無を頻繁に確認するよう設定されています。こうした設定は開発者が迅速に作業するのには便利ですが、本番環境に移行する際にはこの設定を修正しておくべきです。

デフォルトでは、テンプレートがキャッシュに 60ms 残るように設定されています。本番環境で使用する場合、この設定を、例えば 600000ms（10 分）、3600000ms（1 時間）または -1（無期限）に延長してもよいでしょう。

この値を変更するには、システム設定で FreeMarker Engine と Velocity Engine の設定を探します。修正するプロパティは、”resource modification check interval”のラベルが付いているプロパティです。

6 Tomcatでは、DXP web.xmlは$CATALINA_HOME/webapps/ROOT/WEB-INF/web.xmlの中にあります。

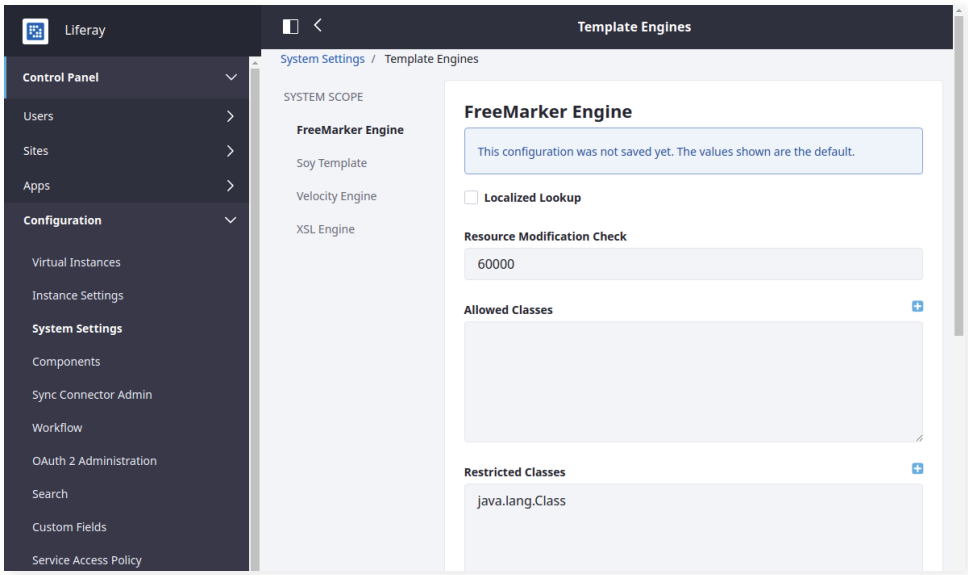


図 6 – Freemarker エンジンのシステム設定

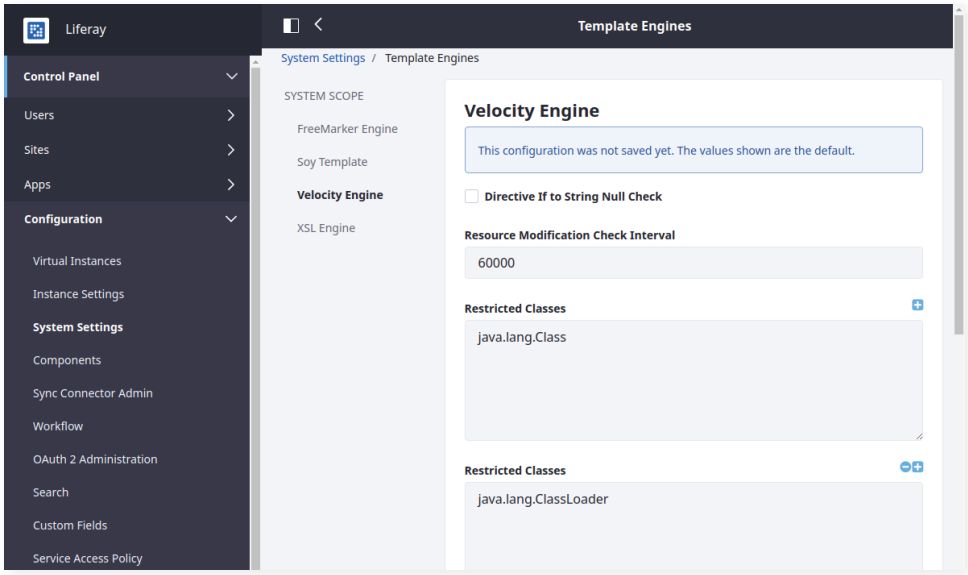


図 7 - Velocity エンジンのシステム設定

## ユーザーセッショントラッカー

Liferay DXP では、管理者は、システムを利用中のユーザーのアクティビティを閲覧できます。この機能はトラブルシューティングには役立つ反面、パフォーマンスの低下につながります。通常は、この機能を無効にすることが推奨されます。

それには、portal-ext.properties に session.tracker.memory.enabled=false を追加します。

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

## ライフレイ・エンタープライズ・サーチ

Liferay DXP は、Elasticsearch エンジンが組み込まれた状態で提供されます（つまり Elasticsearch エンジンは Liferay DXP と同じ JVM で実行されます）。このソリューションには、検索機能を設定不要で使えるというメリットがありますが、本番環境での利用については、ライフレイは公式にサポートしていません。本番環境での利用については、DXP JVM 以外で実行される Elasticsearch のみサポートしています（つまり、Liferay DXP 用の JVM のほかに、Elasticsearch 検索エンジン用に JVM がもう 1 台必要）。

検索エンジンは、キャッシュから大きな恩恵を受けており、検索エンジンの JVM のメモリプロファイルは、コンテンツやウェブの閲覧を目的とした JVM（ライフレイ JVM など）のメモリのプロファイルとは本質的に異なります。このためこの 2 つのアプリケーションは、本番環境では常に切り離しておく必要があります。

以下のセクションでは Elasticsearch 設定の概要について説明しています。デプロイメントの前に、本番環境デプロイメントに関する Elastic のドキュメンテーション「Elasticsearch – the Definitive Guide」を一読されることを強く推奨します。

### デプロイメントのサイズ決定

Elasticsearch のデプロイメントのサイズを決定する際には、CPU、メモリ、ディスクおよびネットワーク容量について十分検討することが必要です。原則として、マシンの台数は増やさないようにしながら、効率的に拡張できるように中型から大型のマシン 7 で Elasticsearch をデプロイするようにしてください。また、同じオペレーティングシステム上で Elasticsearch 用の JVM を複数実行することは避けた方がいいでしょう。

#### CPU

ライフレイでは、Elasticserach エンジンに 8 CPU コア以上を割り当てることを推奨します（ただし、1 台のマシンで Elasticsearch 用の JVM を 1 つだけ実行する場合を想定）。

#### Memory

16GB 以上のメモリを推奨します（理想は 64GB です）。

**Disk**

検索エンジンはインデックスをディスクに格納します。そのため、ディスク I/O 容量が検索のパフォーマンスに大きく影響する可能性があります。可能な場合には、Elasticsearch を SSD 上でデプロイすることを推奨します。SSD を使用できない場合は、高性能ディスク（15,000rpm のドライブ）を推奨します。また、SSD と従来のハードディスクのどちらを使う場合でも、RAID 0 の使用を検討すべきです。

NAS（ネットワーク接続型ストレージ）はネットワークのオーバーヘッドが非常に高くなる可能性があるため Easticsearch には使用しない方がよいでしょう。Amazon Web Services などのパブリッククラウドのインフラを使用している場合は、インスタンスのローカルストレージを使うようにし、Elastic Block Store（EBS）のようなネットワークストレージは避けた方がよいでしょう。

インデックスの総サイズよりも 25% 以上多いディスク容量が必要です。例えば、インデックスを作成すべきデータのサイズが 50GB の場合、少なくとも 65GB の利用可能なディスクスペースを計画する必要があります。インデックスサイズはインデックスが作成されるコンテンツによって異なります。

**クラスタサイズ**

DXP が 1 つまたは 2 つのノードから成る Elasticsearch クラスタで機能するとしても、Elastic が推奨する耐障害性を勘案した最小クラスタサイズは 3 ノードです。

**ネットワーク**

Elasticsearch は、クラスタリングとシャーディングを使用して、高速かつ正確な検索結果を配信します。それには、高速で信頼性の高いネットワークが必要です。最新のデータセンターはマシン間で 1GbE または 10GbE を実現しています。Elasticsearch クラスタを複数のデータセンターに展開することは避けた方がよいでしょう。Elasticsearch は、複数のデータセンターへのデプロイメント（特に、大陸間のように遠く離れた場所にある複数のデータセンターへのデプロイメント）はサポートしていません。

## Elasticsearch の設定

Elasticsearch を起動する前に、elasticsearch.yml に以下のいくつかのプロパティを設定する必要があります。

- cluster.name
    - デフォルトでは、Liferay DXP は、“LiferayElasticsearchCluster” を想定しています。

- 異なるクラスタ名で設定したい場合（例えば“elasticsearch_ production”）は、システム設定で Liferay DXP の Elasticserach の設定も修正する必要があります。ライフレイに設定されたクラスタ名は、Elasticsearch サーバーに設定されたものと一致しなければなりません。

- node.name
  - Elasticsearch クラスタの各ノードには固有の名前を付ける必要があります。
- discovery.zen.ping.unicast.hosts
  - ノードの検出にはマルチキャストよりもユニキャストを使用することを強く推奨します。これによってノードが偶発的にクラスタと結合するのを防ぎます。
  - クラスタ内にあるすべての Elasticsearch のノードが、Elasticsearch の起動時から互いに見える必要はありません。Elasticsearch の起動時にクラスタに接続されていたメンバーの 1 つに新しいノードが接続されると、そのノードは自動的にトポロジー情報を受信し、他のノードと通信します。
- thread_pool.bulk.queue_size
  - ライフレイは、ライフレイと Elasticserach 間のネットワーク呼び出しの回数を減らすため、バルクリクエストを使用します。バルクリクエストのスレッドプール用キューサイズを 100 以上にすることをお勧めします。

上記の設定に加えて、デバッグを目的として、以下を設定することを選択できます

```
# 以下のオーバーヘッド設定はパーセントです。
monitor.jvm.gc.overhead.debug: 40
monitor.jvm.gc.overhead.info: 70
monitor.jvm.gc.overhead.warn: 90
```

## デプロイメントをチューニングする

**JVM**

通常、利用可能なシステムメモリの 45%、最大で 31GB までを Elasticsearch に割り当てる必要があります。通常、Elasticsearch 内ではその他の JVM の設定を変更しないでください。以下の環境変数（ES_HEAP_SIZE）を設定してヒープサイズを設定する必要があります。

Elasticserach サーバーに使用する JVM のベンダーとバージョンは、Liferay DXP で使用しているものと同じである必要があります。

**ファイルシステム**

最低でも 64,000 個のファイル記述子を使用するよう OS を設定する必要があります。Linux では、ファイル記述子はデフォルトで 1024 に設定されています。

Elasticsearch は、NioFS と MMapFA も使用しています。したがって、メモリマップトファイル用に利用可能な仮想メモリを十分に確保することが必要です。

これらの値の設定方法については、システム管理者に確認してください。

## デプロイメントを監視する

Elasticsearch には X-Pack Monitoring という監視ツールが用意されています。X-Pack Monitoring は 2 つのコンポーネントで構成されています。

- パフォーマンスデータを別の Elasticsearch ノードに送信する、Elasticsearch ノードの中にあるサーバー側のコンポーネント
- Elasticsearch パフォーマンス情報の視覚化を可能にする一連の Kibana ダッシュボード

ダッシュボード上で、Elasticsearch クラスタの状態とパフォーマンスを監視することで、検索クラスタのさらに適切なチューニングと管理が行いやすくなります。

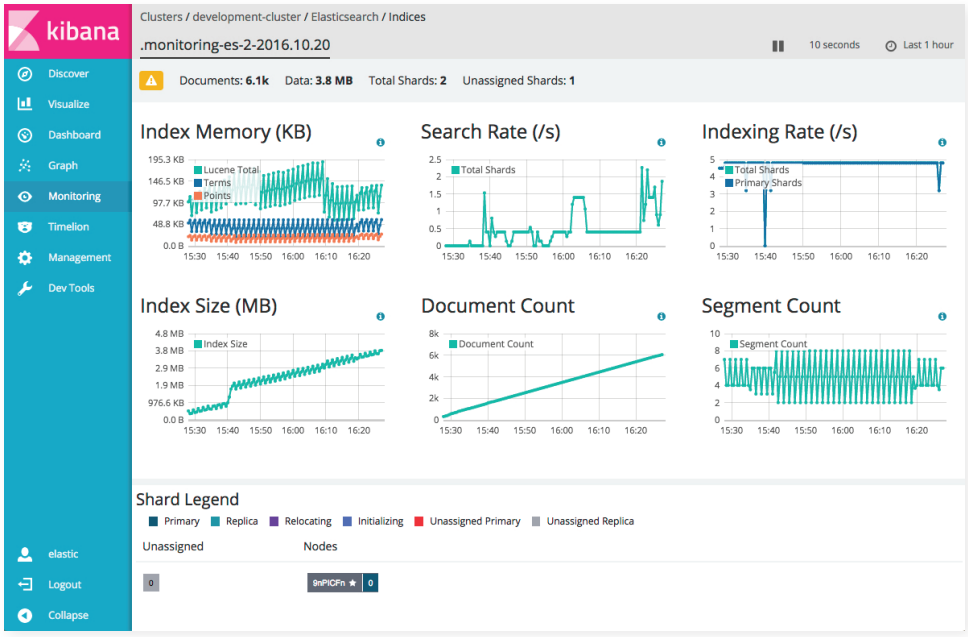


図 8 – X-Pack Monitoring ダッシュボードの表示例

X-Pack Monitoring をライフレイの Elasticsearch クラスタと使用するには、ライフレイのエンタープライズ・サーチサブスクリプションが必要です。詳しくは、営業担当までお問い合わせください。

### デプロイメントを保護する

検索処理の実行に伴い、検索エンジンは大量のデータのインデックスを作成し、そのインデックスにデータを格納します。デフォルトではこのデータへのアクセスは保護されていません。データにアクセスする際にログインが必要なデータベースとは異なり、検索エンジンは格納されている情報の検索やブラウズにログインを必要としません。

Elasticserach のセキュリティのアドオン X-Pack Security を導入すると、ネットワーク経由の暗号化と、検索エンジンアクセス時の認証機能を追加できます。ネットワーク経由の暗号化によって、Elasticsearch ノード間のすべての通信は確実に暗号化され、ライフレイと Elasticsearch サーバー間の通信は保護されます [7]。 検索エンジンアクセス時の認証機能によって、Elasticsearch への接続には、データベースの場合とほぼ同様に認証が必要となります。

これらのセキュリティ機能を使用するには、ライフレイのエンタープライズ・サーチサブスクリプションが必要です。詳しくは、営業担当までお問い合わせください。

## まとめ

上記では、高い耐障害性を備えたライフレイ・デジタルエクスペリエンス・プラットフォームのデプロイメント手順を説明しました。ここで記したアーキテクチャは、将来の成長のための確固たる基盤となるものです。また、ライフレイ・エンジニアリングチームからは、Liferay DXP デプロイメントのチューニングを成功させるためのいくつかの重要な要因が示されました。これらのパラメーターは適用範囲が広く、さまざまな負荷テストシナリオにもパスしてきたものですが、ライフレイエンジニアリングでは、長期にわたりパフォーマンスを維持できるように、Liferay DXP のデプロイメント状況を継続的に監視することを推奨しています。

7 Liferay DXPとElasticsearchクラスタとの通信は、HTTPS/JSON APIではなく、Elasticsearchのバイナリ APIによって行われます。バイナリプロトコルは、HTTP/JSON APIよりもパフォーマンスに優れます。

# 免責事項

ライフレイは、工場出荷状態の製品に対して行われたベンチマーク調査の結果に基づいて、初歩的なチューニングに関するアドバイスしか提供できません。システムを実際に稼働させて活用するシナリオについては、本ドキュメントをお読みになったお客様（システムアーキテクトやビジネスアナリスト）が想定する必要があります。本番稼働に移行する前に、システムに対してお客様の責任でしかるべき負荷テストを行い、カスタムアプリケーションやポートレットに起因する重大なボトルネックおよびシステムとネットワークに関する想定外の問題を明らかにし、適切な設定を実施してください。

# 次のステップ

## ライフレイ・コネクテッドサービス

ライフレイ・コネクテッド・サービス（LCS）では、Liferay DXP ソリューションの管理や監視に役立つ基本性能監視ツールやシステム監視ツールを数多くご用意しています。詳しくは 、liferay.com/liferay-connected-services をご覧ください。

## ライフレイとダイナトレース社

ライフレイは、業界をリードするアプリケーションパフォーマンス管理（APM）ソリューションプロバイダーのダイナトレース（Dynatrace）社と提携しています。ライフレイ向けの Dynatrace と Fastpack を導入することで、Liferay DXP のデプロイメントに関して、より詳細なパフォーマンス関連情報を入手することができます。詳しくは、liferay.com/dynatrace-apm をご覧ください。

## ライフレイ・グローバルサービス

ライフレイ・グローバルサービスには、Liferay DXP Go Live やパフォーマンスチューニングのコンサルテーションを専門とするスペシャリストが所属しています。詳しくは liferay.co.jp/contact-us よりお問い合わせください。

ライフレイは、様々なデバイスを通して Web のデジタル体験を創造するソフトウェアを提供しています。ライフレイのプラットフォームはオープンソースがもたらす革新性と合わせ、高い信頼性とセキュリティを兼ね備えています。我々はビジネスとテクノロジーによって、世界に優れた足跡を残すことを目指し日々活動しています。ライフレイの製品は世界中の有力企業に採用されており、金融から製造、ヘルスケア、行政、保険、小売、フランチャイズなど様々な業界へソリューションを提供しています。詳細は、liferay.co.jp へアクセスしてください。



190709