Liferay DXP profile for Dynatrace

This document provides description of Dynatrace profile created for monitoring of Liferay DXP installations.

Version	Liferay version	Dynatrace Version	Created by	Created on
1	DXP (7.0 ga1)	6.3.7 build 1007	Brian Wilson, Josef Sustacek	Aug 12, 2016

Table of Contents

Business Transactions

The profile contains custom Business Transaction which mainly serve as a way to classify PurePaths (with Web Requests) into various groups based on how Liferay will be handling these PurePaths. Following sections will describe most important transactions.

Most of the Business Transactions use one or more custom Measures.

Portlet - [portlet name] - [portlet phase]

We have identified a group of *core Liferay portlets*, which are most commonly used in Liferay deployments (by end users). We are interested in knowing when a portlet request for one of these portlets has been sent to Liferay and how it was processed.

Most of the portlets use Liferay MVC framework as their implementation and transactions then split on the significant *portlet parameter* which was used to form the portlet URL. For action requests it is the *action name* (*javax.portlet.action*), for resource requests it is the *resource ID* (*p_p_resource_id*). For render requests, there are no splits defined, since most of the time, portlets are rendered with implicit params and it's impossible to setup a measure in Dynatrace with default value returned when not computed for given PurePath.

The Business Transactions are named with the pattern *Portlet - [portlet name] - [portlet phase]*. The third part, *[portlet phase]* covers one of the portlet phases, as declared by JSR 286:

- *action processed* - action phase
- *resource served* - resource phase
- *view rendered* - render phase

Not all selected portlets have Business Transactions for all the portlet phases. Some portlets are not using the missing phases or they are used only sporadically by end users.

Examples of created Business Transactions:

- *Portlet - Asset Publisher - resource served*
- *Portlet - Asset Publisher - view rendered*
- *Portlet - Calendar - action processed*
- *Portlet - Calendar - resource served*
- *Portlet - Calendar - view rendered*
- *Portlet - Search - view rendered*

Business Transactions were defined for these core portlets:

Portlet	**Portlet name (ID)**
Asset Publisher	*com_liferay_asset_publisher_web_portlet_AssetPublisherPortlet*

Blogs	*com_liferay_blogs_web_portlet_BlogsPortlet*
Calendar	*com_liferay_calendar_web_portlet_CalendarPortlet*
Documents and Media	*com_liferay_document_library_web_portlet_DLPortlet*
Media Gallery	*com_liferay_document_library_web_portlet_IGDisplayPortlet*
Message Boards	*com_liferay_message_boards_web_portlet_MBPortlet*
Search	*com_liferay_portal_search_web_portlet_SearchPortlet*
Sign In	*com_liferay_login_web_portlet_LoginPortlet*
Web Content	*com_liferay_journal_web_portlet_JournalPortlet*
Web Content Display	*com_liferay_journal_content_web_portlet_JournalContentPortlet*
Wiki	*com_liferay_wiki_web_portlet_WikiPortlet*
Wiki Display	*com_liferay_wiki_web_portlet_WikiDisplayPortlet*

CMS Page Request

Tags any PurePath containing Web Request which will be handled as a request for Liferay Content Management System page. The typical Liferay CMS page contains a set of portlets and belongs to some Liferay site or organization's site.

Liferay CMS page URIs typically start with */web*, */group* or */user*, but when virtual hosts are used for some Liferay site, these might be shortened to contain just the friendly URL of the page.

Split is made on the *internal* friendly URL of the request, the way Liferay servlets see it. It is always the full (longer) version of the URI, not shortened by a virtual host filter. Also, the split transformation removes any *portlet friendly URL mapper* part if present at the end of the URI. All Liferay's portlet friendly URL mappers are delimited by */-/* in the URI.

For details on virtual hosts and portlet friendly URL mappers, see Liferay documentation or FriendlyURLServlet section.

CMS Page Request by Page Type - [page type]

Each of these Business Transactions matches a subset of PurePaths tagged as CMS Page Request, based on the type of the targeted Liferay group and layout set. We classify the CMS pages into three categories based on this type, which translates into three transactions:

- *CMS Page Request by Page Type - Private Site Page*
  - *private pages* of any Liferay *site*
  - URI pattern: */group/**

- *CMS Page Request by Page Type - Private User Page*
    - *private pages* of any Liferay *user*
    - URI pattern: */user/**
- *CMS Page Request by Page Type - Public Page*
    - *public pages* of any Liferay site or user
    - URI pattern: */web/**

There is no split on these transactions, *CMS Page Request* transaction defines the split and will match every PurePath matched by any of these transactions.

CMS Page Request by Request Type - [portlet phase]

Each of these business transactions matches a subset of PurePaths tagged as CMS Page Request. Given transaction is matched if at least one portlet has entered given portlet phase during processing of the PurePath.

We recognize there portlet phases for CMS page requests, which translates into three transactions:

- *CMS Page Request by Request Type - Process Portlet Action*
    - portlet *action* phase
- *CMS Page Request by Request Type - Render Portlets*
    - portlet *render* phase
- *CMS Page Request by Request Type - Serve Portlet Resource*
    - portlet *resource* phase

There is no split on these transactions, *CMS Page Request* transaction defines the split and will match every PurePath matched by any of these transactions.

Transaction for portlet *event* phase was not created due to lack of its use in core portlets. If needed, it can be added by following the pattern and using the measure on the method `PortletContainerUtil.processEvent()` as needed. Sensor for this method might need to be activated in Liferay Sensors configuration.

Please note that one PurePath can be processing both *action* and *render* phases. This happens when a *portlet action URL* is processed and the executed portlet action method does not send a redirect to render phase. Most core Liferay portlets are sending the redirect, to prevent accidental re-submission of forms. Since portlet events are sent from *action* phase, also *event* phase can be in one PurePath with *action* phase.

Portlet [phase]

Each of these Business Transactions covers execution of one portlet phase (JSR 286) of *any* portlet deployed into Liferay DXP. The portlet does not have to be processed in the context of a CMS page, for example if *ajaxable* is set to *true* for given portlet, it can be rendered on AJAX URI */c/portal/render_portlet* without any theme or layout decorations.

Three Business Transaction belong to this group:

- *Portlet action processed*
- *Portlet view rendered*
- *Portlet resource served*

The transactions are capturing the *portlet container method time* (not whole PurePath time).

The split is done on *portletId* for for all transactions. For instanceable portlets, this includes the *instanceId*.

For the same reason as in CMS Page Request by Request Type - [portlet phase], the transaction for portlet *event* phase was not created, but can be added if necessary.

CSS Client Side

Tags any User Action which involves loading of a CSS file from the server. This could either be a static CSS file (**.css*) or dynamic CSS content assembled by Liferay servlet (e.g. **.jsp* generating a CSS content).

CSS Server Side

Tags any PurePath containing Web Request which is serving a CSS file from server. This could either be a static CSS file (**.css*) or dynamic CSS content assembled by Liferay servlet (e.g. **.jsp* generating a CSS content).

JavaScript Client Side

Tags any User Action which involves loading of a JavaScript file from the server. This could either be a static JavaScript file (**.js*) or dynamic JavaScript content assembled by Liferay servlet (e.g. */combo/* generating a JavaScript content).

JavaScript Server Side

Tags any PurePath containing Web Request which is serving a JavaScript file from server. This could either be a static JavaScript file (**.js*) or dynamic CSS content assembled by Liferay servlet (e.g. */combo/* generating a JavaScript content).

Liferay Login Request (non-SSO)

Tags any PurePath which will be authenticating the user into Liferay using *username* and *password* provided by the user. Uses code instrumentation on Liferay's core class AuthenticatedSessionManagerUtil.login().

This Business Transaction does not have a split on username. Data is stored in Performance Warehouse and the set of distinct usernames could be very large. For one PurePath, the username can be retrieved by drilling down through the call stack of the PurePath.

Liferay Login Request (SSO)

Tags any PurePath containing Web Request which will try to authenticate the user using one of the Liferay SSO handlers or Remember Me cookies. The exact set of handlers depends on Liferay configuration. Uses matching on URI */c/portal/login*.

Please note that Remember Me authentication will also be used on any CMS page (*/web*, */user*, */group*), so not all auto-login requests using Remember Me cookies will be tagged with this Business Transaction.

This Business Transaction also does not have a split on username, for the same reasons as Liferay Login Request (non-SSO) mentioned previously.

Liferay Logout Request

Tags any PurePath containing Web Request which will sign the user out of Liferay DXP. Uses matching on URI */c/portal/logout*, which covers both SSO and non-SSO authenticated users.

This Business Transaction also does not have a split on username, for the same reasons as Liferay Login Request (non-SSO) mentioned previously.

Liferay Signed-in User - Non-UEM

Tags all PurePaths containing Web Request (no filter is specified), provides information about user's identity from Liferay's point of view. Transaction splits on the user attributes (*user ID*, *screen name* and *primary email*) of the user authenticated into Liferay. Split values are fetched from HTTP session attributes. .

Please note that for unauthenticated users, Liferay uses a default user object created for given Liferay instance (not *null* object).

To analyze all transactions for a particular user where Dynatrace UEM is not available, search a user name in this view and drill down to PurePaths.

This Business Transaction is not stored in Performance Warehouse due to split on users, which might generate too many unique values. As a result, data from this transaction will be available in Dynatrace only temporarily.

Liferay WebServer Document Request

Tags any PurePath containing Web Request which will be serving a file (its binary content) from Liferay's Document Library through WebServer servlet.

Matches URIs */documents/**.

Liferay WebServer Image Request

Tags any PurePath containing Web Request which will be serving an image (its binary content) from Liferay through WebServer servlet.

Matches URIs */image/**.

User Action Response Size

Tags every User Action and measures the average size of the transferred data.

Workflow Engine - Kaleo GraphWalker Message Received

Tags every PurePath within which at least one transition was made in the Liferay's Kaleo Workflow Engine. Measures the time it took to process all transitions within given PurePath.

Liferay HTTP API Request

Tags any PurePath containing Web Request which will be handled by one of the Liferay API servlets. These are all mapped to URIs starting with */api*.

Various APIs are available in Liferay, this transaction covers them all: Atom, Axis, JSON, JSON WS, Spring, Liferay.

AJAX Request

Tags any PurePath containing a Web Request which was made using some modern JavaScript library, like AlloyUI or jQuery.

Measures

Liferay profile contains a set of custom measures, which are used either in Business Transactions, dashboards (graphs), incident definitions or to provide additional information when drilling down through one individual PurePath. Following section provides overview of the most important custom measures and their use.

Business Transaction measures

Many measures are used within the custom Business Transactions, either as filtres, splits or calculations.

Portlet measures

As listed in section on Business Transactions, we have a set of transactions which cover the core Liferay portlets. These are backed by a set of measures providing the filters and splits for these transactions.

Measures are relying on the instrumentation sensors, mainly on PortletContainerUtil.

Example measures:

- *Calendar - render request*
- *Documents and Media Display - resource request*
- *Web Content - action request*

Liferay Signed-in User measures

Three measures were created to extract the identity of users logged into Liferay:

- *User in Liferay ('user.getEmailAddress()' from HTTP session)*
- *User in Liferay ('user.getScreenName()' from HTTP session)*
- *User in Liferay ('userId' from HTTP session)*

Each measure produces one type of identification of Liferay user, each is value is unique within all users in one Liferay instance.

Measures are relying on *Servlets Sensor Pack* and its configuration in application server agents, see Application Servers (Tomcat, JBoss).

These measures are used for Business Transaction Liferay Signed-in User - Non-UEM and also to tag *User Visits* in UEM Configuration.

CMS pages measures

Various measures providing filters and splits to categorize Web Request which will be handled by Liferay as Content Management System page requests.

There are two ways how to recognize the CMS page URI pattern: *external* and *internal*. *External* detection relies in the URI as provided by the Web Request. Although this looks like an obvious

choice how to detect CMS page request, due to Liferay's virtual hosts and friendly URL mappings, not all requests may be captured.

Internal detection is more reliable, since it's reading the URI based on instrumentation of Liferay's CMS servlet, after the virtual host filter and friendly URL mapping was performed by Liferay. For details, see com.liferay.portal.servlet.FriendlyURLServlet section.

These measures are used in various CMS transactions, see CMS Page Request, CMS Page Request by Page Type - [page type] and CMS Page Request by Request Type - [portlet phase].

CSS and JavaScript measures

Various measures used to determine if a Web Request is serving CSS or JavaScript content. Liferay contains several servlets and JSPs generating dynamic CSS and JavaScript based on passed parameters.

Measures are used in Business Transactions CSS Client Side, CSS Server Side, JavaScript Client Side and JavaScript Server Side.

Liferay Login Method (non-SSO)

Used as a filter to match Liferay authentications using *username* and *password*, provided by the user, most likely through configured *login* portlet like the core *Sign In* portlet.

This measure uses instrumentation on method AuthenticatedSessionManagerUtil.login(), since this method has to be (by Liferay contract) used by any portlet which authenticates user using explicit credentials (username and password).

Measure is used by transaction Liferay Login Request (non-SSO).

Liferay Login URI (SSO)

Used as a filter to match Liferay SSO authentications, which all occur on URI */c/portal/login*. All Liferay SSO filters are mapped to this URI, together with *Remember Me* autologin filter.

If user can be logged in using one of the filters, redirect is sent to user's landing page. If no filter can authenticate the user, redirect is sent to the page which will render the configured *login* portlet. This typically is the default, *Sign In* portlet, but can be changed using Liferay configuration.

Please note that *Remember Me* is also mapped to all CMS pages (*/web/**, */user/**, */group/**), so not all auto-logins via *Remember Me* cookies will go through */c/portal/login*.

Measure is used by transaction Liferay Login Request (SSO).

Liferay Logout Request

Used as a filter to match Liferay DXP logouts on URI */c/portal/logout*. Both manually and SSO authenticated users are using this URI to log out of Liferay DXP.

Measure is used by transaction Liferay Logout request .

Liferay WebServer Documents URI

Used as a filter on Web Requests which will be handled by Liferay Web Server servlet.

Matches URIs starting with */documents/,* used to serve Document Library documents and their thumbnails, for example in *Documents and Media* portlet.

Measure is used in Business Transaction Liferay WebServer Document Request.

Liferay WebServer Images URI

Used as a filter on Web Requests which will be handled by Liferay Web Server servlet.

Matches URIs starting with */image/,* used to serve dynamically stored images, like user account's pictures.

Measure is used in Business Transaction Liferay WebServer Image Request.

XHR by X-Requested-With header

Provides a way to determine if a Web Request should be categorized as a regular or AJAX request.

The measure matches Web Requests on HTTP header *X-Requested-With* with value *XMLHttpRequest*. All modern frameworks (YUI, AlloyUI, jQuery) add this header when making an AJAX call.

Measure is used in Business Transaction AJAX Request.

Liferay - EhCache - [cache name] - [cache metric]

A set of measures to capture statistics of Liferay EhCache caches through JMX. Can be found in *Measures -> Server Side Performance -> Agent based Measures -> Custom JMX*.

For every cache, we are capturing three metrics and calculating fourth:

- *MemoryStoreObjectCount* - number of elements in cache
- *MaxElementsInMemory* - maximal size of the cache
- *InMemoryHitPercentage* - rate of success for cache element being present in cache when requested
- *cache occupancy rate (in-memory)* - calculated from first and second metric

Following caches are covered by these measures:

- *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.RoleImpl*
- *com.liferay.portal.kernel.dao.orm.EntityCache.com.liferay.portal.model.impl.UserGroupRoleImpl*
- *com.liferay.portal.kernel.dao.orm.EntityCache.com.liferay.portal.model.impl.UserImpl*
- *com.liferay.portal.security.permission.PermissionCacheUtil_PERMISSION*
- *com.liferay.portal.security.permission.PermissionCacheUtil_PERMISSION_CHECKER_BAG*
- *com.liferay.portal.security.permission.PermissionCacheUtil_RESOURCE_BLOCK_IDS_BAG*

Dynatrace automatically fetches new value for each JMX metric every 10 seconds, using configured MBean definitions.

Liferay utilizes a large number of caches (300+), these measures only capture basic set of essential caches. Measures for additional caches can be added following provided pattern, if necessary.

Measures are used in Liferay Ehcache Metrics dashboard and related views.

Database pool measures

Although Liferay uses one database schema to store its relational data, two data sources are used to query the database: main *Liferay data source* and *counter data source*. The use of two data sources is necessary to prevent deadlocks (see comment on property *counter.jdbc.prefix* inside *portal.properties* for details).

Liferay will by default create its own internal data sources, with all connections pooled for better performance, or it can get them as a JNDI resource from the application server. Liferay will then rely on application server's configuration - that the retrieved data sources will be defined with proper connection pooling.

Following measures cover the most commonly used pool implementations which can be used in Liferay. All measures can be found in *Server Side Measures -> Agent based Measures*, in a category denoting in which application server will data for this measure be available.

Custom JMX category

Liferay - JDBC Pool - C3p0, All PooledDataSource beans (sum) - [metric]

A set of measures reporting metrics for all C3p0 pools present in the JVM environment. If Liferay is using C3p0 pool of database connections (either as internal pool or through JNDI) and there is no other pool of this type defined, the measure will return data for the Liferay connection pools.

If there are more C3p0 pools, the data will be aggregated. C3p0 generates unique identifier for every pool after its start, not related to the the pool's resource name (the JNDI name under which it's published in the application server), so it's not possible to target only the Liferay pools.

Liferay - JDBC Pool - HikariCP, All Pool beans (sum) - [metric]
A set of measures reporting metrics for all Hikari pools present in the JVM environment, typically created internally by Liferay.

If there are more Hikari pools than the Liferay ones, the data will be aggregated. Hikari generates unique identifier for every pool after its start, not related to the the pool's logical name, so it's not possible to target only the Liferay pools.

Liferay - JDBC Pool - TomcatJdbcPool, All Pool beans (sum) - [metric]
A set of measures reporting metrics for all Tomcat pools created by Liferay internally. This metric is exposed by Liferay itself, so no data from other pools should be reported in this measures.

Tomcat category

JDBC Pool - TomcatJdbcPool "jdbc/LiferayPool" - [metric]
A set of measures reporting metrics for a Tomcat pool named *"jdbc/LiferayPool"*, which is the default JNDI name of the external Liferay pool as used in Liferay bundles.

If your Liferay installation is using external Tomcat pool, but it's named differently, you will have to update the measures accordingly or create a set of your own measures mirroring the existing ones.

JDBC Pool - TomcatJdbcPool "jdbc/LiferayCounterPool" - [metric]
Same as previous, only reporting measures for a differently named JNDI resource, allocated for Liferay's *counter data source*.

JBoss category

JDBC Pool - ManagedConnectionPool, all Pool beans (sum) - [metric]
A set of measures to get current number of busy connections and maximal size of the pool, together with the rate between the two.

Please note that JBoss publishes only a very limited database pools data into JMX (if any), so the measures might not report valid information.

Thread pools measures

Liferay fully relies on its application server to provide the thread pools for HTTP / AJP connectors, which are processing all WebRequests. The naming and configuration of the pools depends on the application server settings, the following measures are reflecting the default setup and naming as used in Liferay bundles.

All measures can be found in *Server Side Measures -> Agent based Measures*, in a category denoting in which application server will data for this measure be available.

Tomcat

ThreadPool - Connector "ajp-nio-8009" - [metric]
These measures captures the number of busy threads and maximal size of the pool, together with rate between the two, for the default Tomcat AJP NIO connector listening on port 8009.

If your AJP connector is configured differently (different port number, using BIO, APR...) the name of the JMX bean might be different and you might need to update the measures accordingly. See Tomcat documentation for details.

ThreadPool - Connector "http-nio-8080" - [metric]
These measures captures the number of busy threads and maximal size of the pool, together with the rate between the two, for the default Tomcat HTTP NIO connector, listening on port 8080.

If your HTTP connector is configured differently (different port number, using BIO, APR...) the name of the JMX bean might be different and you might need to update the measures accordingly. See Tomcat documentation for details.

JBoss

ThreadPool - all pools (sum) - [metric]
These measures provide the number of busy and total threads in JBoss thread pools.

Please note that JBoss publishes only a very limited thread pools data into JMX (if any), so the measures might not report valid information.

Method execution time

We've identified several places in Liferay, where we want to capture precise time it takes to execute particular method. These measures are then used either in calculations for Business Transactions or in dashboards. We capture the total execution time for each method, the method's time itself including any sub-calls the method makes.

All these measures rely on correctly enabled sensors, see Java Code Instrumentation section.

PortletContainerUtil.[method name] Time

Whenever a portlet request has to be processed by Liferay, static utility class `PortletContainerUtil` is responsible for the whole execution. Concrete method is chosen based on the portlet phase which the portlet request belongs to.

We're capturing time of following methods:

- *PortletContainerUtil.processAction()*

- *PortletContainerUtil.processEvent()*
- *PortletContainerUtil.render()*
- *PortletContainerUtil.serveResource()*

All the measures rely on the sensors as outlined in instrumentation section, chapter com.liferay.portal.kernel.portlet.PortletContainerUtil.

Please note that `processEvent()` does not have its sensor active by default. If you want to capture time of portlet events processing, enable the sensor for this method.

Liferay Search Engine - [method name] Time

A set of measures capturing the execution time in several places of Liferay Search API. Covers possible Search Engine implementations as supported by Liferay (Elasticsearch or Solr).

Following measures were created:

- *Liferay Search Engine - FacetedSearcherImpl.search() Time*
  - method called by *Search* portlet (*portletId= com_liferay_portal_search_web_portlet_SearchPortlet*)
  - delegates to `IndexSearcherHelperUtil.search()`
- *Liferay Search Engine - IndexSearcherHelperUtil.search() Time*
  - delegates to configured Search Engine implementation (Elasticsearch or Solr)
- *Liferay Search Engine - ElasticsearchIndexSearcher.search() Time*
  - method called by `IndexSearcherHelperUtil` if Liferay is configured to use Elasticsearch
- *Liferay Search Engine - SolrIndexSearcher.search() Time*
  - method called by `IndexSearcherHelperUtil` if Liferay is configured to use Solr

Measures rely on the sensors as outlined in the instrumentation section, chapter FacetedSearcherImpl and onward.

Agent Sensors Packs and their configuration

Some measures in Liferay profile rely on Dynatrace Sensors Packs and their configuration for connected agents, following section lists the changes from defaults.

Application Servers (Tomcat, JBoss)

Liferay runs on variety of Java application servers, appropriate Dynatrace application server agent has to have the *Servlets* Sensor Pack enabled, which is true by default.

Also, in configuration of this Sensor Pack, following session attributes have to be captured by Dynatrace, since they are used in Liferay Signed-in User measures and in UEM Configuration:

- session attribute *USER* with accessor `getEmailAddress()`
- session attribute *USER* with accessor `getScreenName()`
- session attribute *USER_ID* with no accessor

Sessions attributes do not have to be captured in all filters and servlets, just on the entry points, so the checkbox *Capture details in all filters and servlets* can be left unchecked (if not needed for other purposes).

UEM Configuration

All User Visits are tagged by one of the measures providing identity of the users signed into Liferay. These measures rely on *Servlets* Sensor Pack and its configuration in application server agents, for details see Liferay Signed-in User measures or sensors configuration for Application Servers (Tomcat, JBoss).

One of the following measures should be selected to tag the user visits:

- *User in Liferay ('user.getEmailAddress()' from HTTP session)* (default)
    - will tag visits using the *primary email address* from the user's account in Liferay
    - emails can be chosen by each user, imported from external system (LDAP) or automatically generated, depending on Liferay's configuration
    - *primary email* is always unique in one Liferay instance
- *User in Liferay ('user.getScreenName()' from HTTP session)*
    - will tag visits with *screen name* from the user's account in Liferay
    - *screen name* is either chosen by each user, imported from external system (LDAP) or automatically generated, depending on Liferay's configuration
    - *screen name* is always unique in one Liferay instance
- *User in Liferay ('userId' from HTTP session)*
    - will tag visits with *userId* of user's account in Liferay
    - *userId* is a natural number generated by Liferay when a user account is created
    - *userId* is always unique in one Liferay instance

The measure used for UEM visits can be changed in *Liferay profile details -> User Experience -> Default Applications* (or selected one) -> *General / Tag visits with*.

Java Code Instrumentation (Sensors)

Many custom measures in Liferay profile are depending on instrumentation of methods in various places or Liferay code base. All Liferay related bytecode changes are defined in *Sensor Group* named *Liferay*. Following is the list of classes and methods which are actively instrumented.

Sensor Group also contain a few inactive sensors, which are not needed by default, but can be activated if related measures are needed.

com.liferay.portal.kernel.portlet.PortletContainerUtil

This class is used to handle all portlet requests in Liferay. It delegates the execution to the portlet implementation class and its JSR 286 methods.

`PortletContainer` class follows the standard Liferay pattern `XUtil` (static methods facade) + `X` (interface) + `Ximpl`.

All instrumented methods capture the third argument, to get the information which portlet instance is being rendered / is processing action / is serving portlet resource. Sensors capture `liferayPortlet.getPortletId()` from the third argument, which will be either:

- the *portlet name* from *portlet.xml* (or the OSGi annotation) for non-instanceable portlets
  - example: *com_liferay_login_web_portlet_LoginPortlet*
- the *portlet name* + *instanceId* for instanceable portlets
  - example: *com_liferay_wiki_web_portlet_WikiDisplayPortlet_INSTANCE_4bLTfXEXlQNs*

render()

`void render(httpRequest, httpResponse, liferayPortlet)` handles portlet *render phase* requests. Method will be called for every portlet instance which is being rendered in Liferay portlet container. Portlet can be rendered either on server side as part of a CMS page request or using AJAX for ajaxable portlets, on URI */c/portal/render_portlet*.

Method can be invoked once or multiple times for every PurePath, depending on how many portlets need to be rendered - one portlet render URL can trigger rendering of multiple portlets, depending on portlet window states.

Every method invocation within one PurePath will have a unique portlet instance object, with unique *portletId* being captured (see above).

processAction()

`void processAction(httpRequest, httpResponse, liferayPortlet)` handles portlet *action phase* requests. Method will be called whenever a portlet action URL is being processed by Liferay.

Method is invoked at most once every PurePath, for the portlet instance to which the action URL belongs.

serveResource()

`void serveResource(httpRequest, httpResponse, liferayPortlet)` handles portlet *resource phase* requests. Method will be called whenever a portlet resource URL is being processed by Liferay.

Method is invoked at most once every PurePath, for the portlet instance to which the resource URL belongs.

com.liferay.portal.kernel.security.auth.session.AuthenticatedSessionManagerUtil

login()

`void login(httpRequest, httpResponse, String userName, String password, boolean rememberMe, String authType)` is the main method used by Liferay to authenticate users using *username* and *password* (non-SSO authentication). It is called from the stock *Sigh In* portlet and should also be used by any other custom login portlet, which would be used as a way of authenticating users into Liferay.

Sensor is capturing third argument, *username*. Sensor does *not* capture *password* (fourth argument), since it's passed to this method in plain text.

com.liferay.portal.workflow.kaleo.runtime.internal.graph.messaging.PathElementMessageListener

Listener class from *kaleo-web* plugin which handles all messages sent to destination *liferay/kaleo_graph_walker* on Liferay Message Bus. Messages are sent to this destination whenever there's a need to make a transition in one of the workflow instances.

Listener asynchronously receives the message and executes required transition, together with sending additional messages, if the transition results into one or more transitions to be made.

doReceive()

`void doReceive(message)` receives the message with the description of the transition to be made and executes it. Argument is not captured.

This method starts a new PurePath if necessary, since the workflow engine is fully asynchronous. Transitions can happen without any user interaction, for example as a result of a timer event.

com.liferay.portal.servlet.FriendlyURLServlet

This servlet handles all requests to Liferay CMS pages. It is mapped in Liferay to handle patterns */web/**, */group/** and */user/** (see *liferay-web.xml*). We instrument this servlet to be able to capture all requests which will be handled as Liferay CMS page request.

It's important for Dynatrace monitoring to filter CMS URIs using this method and not just externally on Web Request and the URI that gets passed in the HTTP requests (*external* URI). The external URI may not contain any of the mappings mentioned above and neither the site friendly URL key (second part of the URI). This may be supplied by one of Liferay filters, for example based on a virtual host mapping configured for Liferay sites.

Examples of *external* vs. *internal* URIs:

- *http://hr.company.com*
  - external URI is empty
  - internal URI will be */web/hr* if there is a site with friendly URL *hr* which has its public pages mapped to a virtual host *hr.company.com*
- *http://customers.company.com/home*
  - external URI is */home*
  - internal URI will be */group/customers/home* if there is a site with friendly URL */customers* which has its private pages mapped to a virtual host *customers.company.com* and there is a private page */home*
- *http://johndoe.com/welcome*
  - external URI is */welcome*
  - internal URI will be */user/john.doe/welcome* if there is a user's site with friendly URL *john.doe* which has its private pages mapped to a virtual host *johndoe.com* and contains a page */welcome*
- *https://www.liferay.com/web/guest/home*
  - external URI is */web/guest/home*
  - internal URI is the same as external, since it starts with one of the recognized static patterns (*/web*)

service()

`void service(httpRequest, httpResponse)` is the standard Servlet API method, which gets invoked for every HTTP request served by this servlet. First argument is captured with accessor `httpRequest.getRequestURI()`, which gives us the complete *internal URI* the way Liferay application sees it, after any internal forwards or rewrite rules have been applied by Liferay filters.

The value captured from first argument will typically be:

- */web/guest*
  - request to site *guest* without specifying the page, just telling Liferay we want to fetch public page

    - results into the first available page (from site *guest*) being rendered
- */web/guest/blogs*
    - request to public page *blogs* in site *guest*
- */user/john.doe/welcome*
    - request for private user page *welcome* owned by user *john.doe*
- */group/hr/benefits/-/blogs/how-to-get-benefits*
    - request for private site page *benefits* in site *hr, with portlet friendly URL mapping /-/blogs/how-to-get-benefits*.

com.liferay.portal.search.facet.internal.faceted.searcher.FacetedSearcherImpl

This class is used by *Search* portlet to return the documents matching the keywords and other search inputs (facets values).

doSearch()

`Hits doSearch(searchContext)` is used to form a search query based on the inputs provided in *Search* portlet. Method delegates to `IndexSearcherHelperUtil.search()` .

com.liferay.portal.kernel.search.IndexSearcherHelperUtil

Static utility class, determines which search engine was configured in Liferay (default Elasticsearch or Solr) and delegates the search to appropriate implementation of selected search engine.

search()

`Hits search(searchContext, query)` is called from `FacetedSearcherImpl` to get search results from search engine configured in Liferay.

com.liferay.portal.search.elasticsearch.internal.ElasticsearchIndexSearcher

Implementation of Liferay search engine using Elasticsearch (ES). It comes with Liferay out of the box and is enabled until another search engine plugin is deployed into Liferay (like *solr-web*).

search()

`Hits search(searchContext, query)` is returning the matching documents from embedded ES (or external ES server, if configured), using ES binary API to issue the search request and get results.

com.liferay.portal.search.solr.internal.SolrIndexSearcher

Implementation of Liferay search engine using Solr. Solr integration will be enabled in Liferay only after Solr plugin (*solr-web*) is deployed into Liferay.

Liferay uses Solr HTTP API to query Solr, since the search servers are typically not running on the same VM as the application server nodes with Liferay.

search()

`Hits search(searchContext, query)` is returning the matching documents from Solr, using Solr HTTP API to issue the search request and get results.

Dashboards

For details about the individual Dashboards in the fastpack, please refer to the Dynatrace Liferay Fastpack page.

Resources

1. Portlet Specification 2.0 (JSR 286): https://jcp.org/en/jsr/detail?id=286
2. Understanding the Java Portlet Specification 2.0 (JSR 286): http://www.oracle.com/technetwork/java/jsr286-141866.html
3. Liferay Digital Experience Platform (DXP): https://www.liferay.com/digital-experience-platform
4. Liferay Developer Network: https://dev.liferay.com
    - User & Admin: https://dev.liferay.com/develop/tutorials?p_p_id=2_WAR_knowledgebaseportlet&p_p_lifecycle=0&_2_WAR_knowledgebaseportlet_kbFolderUrlTitle=7-0
    - Developer: https://dev.liferay.com/develop/tutorials
5. Tomcat 8.0 Connectors:
    - https://tomcat.apache.org/tomcat-8.0-doc/connectors.html
    - https://tomcat.apache.org/tomcat-8.0-doc/config/ajp.html
    - https://tomcat.apache.org/tomcat-8.0-doc/config/http.html
6. JBoss Enterprise Application platform 6.4: Administration and Configuration Guide: https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.4/html-single/Administration_and_Configuration_Guide/