Liferay profile for Dynatrace

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

Version	Liferay version	Dynatrace Version	Created by	Created on
1	6.2 EE (sp10)	6.1.0 build 7888	Brian Wilson, Josef Sustacek	Feb 27, 2015
1.01	6.2 EE (sp10)	6.1.0 build 7888	Brian Wilson, Josef Sustacek	Mar 2, 2015

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 these portlets has been sent to Liferay and how it was processed.

Most of the portlets use Liferay Struts framework as its implementation and transactions then split on the *Struts action path* which was passed to the Struts portlet processor. The only exception is *Calendar* portlet, see details below.

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

- *basic action processed* - action phase
- *basic resource served* - resource phase
- *basic 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. Word *basic* was used to indicate that only a subset of available Struts actions paths is covered, not the whole set. The reason is to be able to use Dynatrace's baselining on these Business Transactions, automatically alerting if performance of these transactions degrades.

The pattern has one exception - *Calendar* portlet. Transactions for this portlet are lacking the word *basic*, because this portlet is not implemented as a `StrutsPortlet`, so we cannot create a subset of actions for given phase. For this reason, we are capturing *all* portlet requests for given portlet phase.

Examples of created Business Transactions:

- *Portlet - Asset Publisher - basic view rendered*
- *Portlet - Calendar - action processed*
- *Portlet - Calendar - portlet rendered*
- *Portlet - Calendar - resource served*
- *Portlet - Documents and Media - basic action processed*
- *Portlet - Documents and Media - basic resource served*
- *Portlet - Documents and Media - basic view rendered*
- *Portlet - Sign In - basic action processed*

- *Portlet - Sign In - basic view rendered*

Business Transactions were defined for these core portlets, with corresponding *portletId* in the brackets:

- *Asset Publisher* (101)
- *Blogs* (33)
- *Calendar* (1 from *calendar-portlet* plugin)
- *Documents and Media* (20)
- *Documents and Media Display* (110)
- *Media Gallery* (31)
- *Message Boards* (19)
- *Search* (3)
- *Sign In* (58)
- *Web Content* (15)
- *Web Content Display* (56)
- *Wiki* (36)
- *Wiki Display* (54)

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 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 resubmitting 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 portal. 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 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.

StrutsPortlet [phase]

These Business Transactions are a specialized version of the Portlet [phase] ones, useful for portlets implemented with the use of Liferay Struts Portlet class. Large group of Liferay core portlets is implemented using Struts architecture (Blogs, Message Boards, Web Content or Sign In to name a few).

Three Business Transactions belong to this group:

- *StrutsPortlet action processed*
- *StrutsPortlet rendered*
- *StrutsPortlet resource served*

`StrutsPortlet` class uses parameter *struts_action* to find Struts *action class*, which determines the way how rendering / action processing / resource serving will be handled. We are using value of this parameter to to distinguish distinct actions within one portlet and to do a split on these transactions. Each action may have different complexity, requiring less or more server resources to complete, and mixing these together would mean loosing accuracy of the captured data.

The transactions are capturing the *portlet container method time* (not whole PurePath time), the same way as the Portlet [phase] transactions.

As mentioned above, the split is done on the Struts Path used within given portlet phase. On top of that, Business Transaction *StrutsPortlet rendered* uses additional split on *portletId* (for instanceable portlets, this includes the *instanceId*). Although all Struts Paths are prefixed by unique key based on portlet type (like */journal/* or */document_library/*), there can be multiple portlet instances of the same portlet (within one PurePath, on one CMS page), each rendering the same Struts Path.

If we did not have the split on *portletId*, all the renderings with the same Struts Path would be counted only as one (not captured multiple times for one PurePath). Good example is the

Web Content Display portlet, which is instanceable and most often renders Struts path */journal_display/view* in all the instances for one CMS page request. Thanks to the additional splittings, all renderings get captured.

Following is the example of recorded values for all 3 transactions:

- (render) request to */web/guest/web-content-page*:
  - *StrutsPortlet rendered [56_INSTANCE_tK11SIQsxOxw;/journal_content/view]*
  - *StrutsPortlet rendered [56_INSTANCE_fXvIa74jKoUi;/journal_content/view]*
  - *StrutsPortlet rendered [56_INSTANCE_rriFmiqthCfc;/journal_content/view]*
- action request to portlet *33* on page */web/hr/blogs*:
  - *StrutsPortlet action processed [/blogs/view_entry]*
- resource request to portlet 20 on page */group/hr/documents*:
  - *StrutsPortlet resource served [/document_library/view]*

StrutsPortlet action processed and *StrutsPortlet resource served* do not have additional split on *portletId*, because within one PurePath, there can only be one portlet action processed or portlet resource served. Not using the second splitting reduces the number of unique values for these Business Transactions.

For the same reason as in CMS Page Request by Request Type - [portlet phase], the transaction for portlet *event* phase was created, but not enabled by default.

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.

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.

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.

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.

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 LoginUtil.login().

This Business Transaction does not have a split on username. Data is stored in Performance Warehouse, 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 portal. 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 Web Request, which was made by 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 both at the same time.

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 and PortletRequestProcessor from Liferay Struts.

Example measures:

- *Calendar - resource request*
- *Documents and Media Display Render Request - basic Struts Paths*
- *Web Content Action Request - basic Struts Paths*

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 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 LoginUtil.login(), since this method (by Liferay contract) has to be used by any portlet which authenticates user using explicit credentials.

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 one, *Sign In* with portletId 58, 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 portal logouts on URI */c/portal/logout*. Both manually and SSO authenticated users are using this URI to log out of Liferay portal.

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.

EhCache [cache name] - [cache metric]

A set of measures to capture statistics of Liferay EhCache caches through 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

Liferay uses one database schema for its operation, with all connections pooled for better performance. Liferay can either use its own internal connections pool or get the connection as a JNDI resource from application server and rely on it to pool all the connections retrieved from the provided resource.

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*. *C3p0* and *Tomcat Pool* measures are under *Custom JMX* category, since these pools can be used by Liferay in any application server as internal pool. *JBoss Managed Connection Pool* is listed under *JBoss*.

DB Pool - C3p0, All PooledDataSource beans - [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 pool.

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 pool.

DB Pool - Tomcat Pool, "jdbc/LiferayPool" - [metric]

A set of measures reporting metrics for a Tomcat pool named *"jdbc/LiferayPool"*, which is the default JNDI name of the Liferay pool as used in Liferay bundles.

If your Liferay installation is using Tomcat pool, but it's named differently, you will have to update the measures accordingly.

JBoss Managed Connection Pool Connections [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, 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 Web Requests. 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*, under *Tomcat* and *JBoss* categories.

"ajp-bio-8009" - [metric]

These measures captures the number of busy threads and size of the pool, together with rate between the two, for the default Tomcat AJP BIO connector listening on port 8009.

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

"http-bio-8080" - [metric]

These measures captures the number of busy threads and size of the pool, together with the rate between the two, for the default Tomcat HTTP BIO connector, listening on port 8080.

If your HTTP connector is configured differently (different port number, using NIO, 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 [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, so the measures might not report valid information.

Method execution time

We've identified few 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 few places of Liferay Search API. Covers possible Search Engine implementations as supported by Liferay (Lucene or Solr).

Following measures were created:

- *Liferay Search Engine - FacetedSearcher.search() Time*
  - method called by Search portlet (*portletId*= 3)
  - delegates to `SearchEngineUtil.search()`
- *Liferay Search Engine - SearchEngineUtil.search() Time*
  - delegates to configured Search Engine implementation class (Lucene or Solr)
- *Liferay Search Engine - Lucene.search() Time*
  - method called by `SearchEngineUtil` if Liferay is configured to use Lucene
- *Liferay Search Engine - Solr.search() Time*
  - method called by `SearchEngineUtil` if Liferay is configured to use Solr
- *Liferay Search Engine - Elasticsearch.search() Time*
  - method called by `SearchEngineUtil` if Liferay is configured to use Elesticsearch
  - not available in Liferay 6.2, reserved for future use (Liferay 7.0)

Measures rely on the sensors as outlined in the instrumentation section, chapter FacetedSearcher and onward. Please note that *Elasticsearch* does not have its sensor active, since plugin for Elasticsearch integration is not available for Liferay 6.2.

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 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 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.struts.PortletRequestProcessor

This core Liferay class handles distribution of Struts portlet requests to Struts action classes. It is used only for portlets which are implemented as Struts portlets (<portlet-class> is equal to or extends `com.liferay.portlet.StrutsPortlet`). This class passes all handling of the portlet requests to methods in `PortletRequestProcessor` which determines how to handle the request, based on Struts configuration in *struts-config.xml*.

processPath()

`String processPath(httpRequest, httpResponse)` is called on processor to determined which Struts path should be used. Struts path is the return value of the method. Method checks two sources until not-blank value is found:

1. the value of param *struts_action* from the HTTP request;
2. the value in request attribute:
    - used as default, when no explicit path is passed in the portlet request
    - set based on portlet init parameters
    - *portlet-custom.xml -> portlet ->* init params like *view-action* / *edit-action* / *config-action* etc.

Method is called for both *render* and *resource* requests, so it's used in Dynatrace profile to get the Struts path for *portlet render* and *portlet resource* measures.

The method itself does not handle the requests processing, it only determines the path. Methods `process(renderRequest, renderResponse)` and `process(resourceRequest, resourceResponse)` are the main ones where time is spent.

process()

`void process(portletActionRequest, portletActionResponse, String strutsPath)` is called to process *portlet action request*. Third parameter is the Struts action path to be executed. Value comes from portlet parameter *struts_action* retrieved by method `StrutsPortlet.processAction()`.

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 />* value from *portlet.xml* for non-instanceable portlets
    - examples: *36*, *_1_WAR_calendarportlet*
- the *<portlet-name />* + *instanceId* for instanceable portlets
    - examples: *_1_WAR_webformportlet_INSTANCE_tK11SIQsxOxw, 56_INSTANCE_fXvIa74jKoUi*

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 render 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.portlet.login.util.LoginUtil

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.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 as `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.kernel.search.FacetedSearcher

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

search()

`Hits search(searchContext)` is used to form a search query based on the inputs provided by *Search* portlet. This method delegates to `SearchEngineUtil.search()`.

com.liferay.portal.kernel.search.SearchEngineUtil

Static utility class, determines which search engine was configured in Liferay (default Lucene or Solr by deploying *solr-web* plugin) and delegates the search to appropriate implementation of selected search engine.

search()

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

com.liferay.portal.search.lucene.LuceneIndexSearcher

Implementation of Liferay search engine using Lucene. If 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 Lucene, using Lucene Java API to issue the search request and get results.

com.liferay.portal.search.solr.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, the search servers are typically not running on the same VMs 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

Following is a description of custom dashboards created for the purposes of Liferay portal monitoring. They are presenting the stock Dynatrace data together with custom Business Transactions and Measures.

Liferay Overview

This dashboard provides an application health and performance overview.

Incident Alerts for baseline violations of the Portlet Business Transactions are on the left:

- when an incident occurs, right click to drill down to incident details.
- For *CMS Pages*, when an incident occurs, additionally right click to open incident period in either *Liferay CMS Overview* or Liferay CMS Page Performance Analysis dashboard.

Average Response Time for all Portlet Business Transactions and *Count of all Web Requests* on the system are placed on the right.

Layer Breakdown identifies in which APIs Liferay is spending execution time.

Database Time vs. Count plots Total time spent in database interactions vs. the count of database executions. Right click to open in Liferay Database Performance dashboard.

Ehcache In-Memory Occupancy Rate:

- Taken from the Liferay Ehcache Metrics dashboard, this measure is a percent calculation of *MemoryStoredObjectCount* over *MaxElementsInMemory*
- Right Click to open in Liferay Ehcache Metrics dashboard.

Agent Breakdown displays statistics about performance of and how time is consumed in each instrumented application node.

Liferay [portlet type] Overview

These dashboards show a more detailed view of the performance of the portlet business transactions.

On the top, there are three incident alert lights covering the portlet baseline performance as well as out of the box incidents for Process and Host Health. If more incidents are created along these areas, they can easily be added to the criteria.

Business Transaction Hotspots displays the portlets by their split values, including status, failure rate, average, 95th percentile, count and incidents. The data is showed in the context of today vs. yesterday. Right click any line to drill down to more detail.

The graphs at the bottom show the performance for each of the splits and portlet phases in both a percentile view as well as an average vs. count view. The Percentile view only splits by phase while the average vs. count view splits by basic Struts path.

Liferay Search Overview

In addition to the standard Overview graphs, the Liferay Search Overview contains 2 additional dashlets addressing time spent in the search portion of the code, whether it's Solr, Lucene, Elasticsearch or Liferay Search.

- Search Method Time vs. Search Transaction Time
    - This dual axis graph plots the average time of search transactions against the average time spent in the search method and it's children. The Search measures are based on the following methods
        - elasticsearch.search()
        - FacetedSearcher.search()
        - Lucene.search()
        - SearchEngineUtil.search()
        - Solr.search()
- Search API Breakdown
    - This dashlet tracks the time spent in custom defined API's based on the above methods. This dashlet is helpful in identifying when search is spending time in ways it should not. Time is broken down in the following dimensions
        - Execution Average - average total time in a search API
        - Execution Sum - total time spent in search API for the dashboard time period. This covers all sub-dimensions below:
            - CPU Total - the amount of time the search code consumes CPU cycles
            - Sync Sum - the amount of time the code is stuck in a blocked thread
            - Wait Sum - the amount of time the code is stuck in thread wait state
            - Suspension Sum - the amount of time search code is held up by garbage collection
            - I/O - this time must be inferred as it's not a direct measure. Whatever Execution Sum that is not covered under CPU, Sync, Wait and Suspension is I/O time.
    - Under ideal circumstances, Execution time should be spent in CPU and I/O. An increase of Wait, Sync and Suspension time could indicate the system is encountering a performance issue.

Liferay CMS Page Performance Analysis

This is a parent/child dashboard. Select one or more page splits in the *CMS Pages* breakdown in the upper left dashlet and the other dashlets will auto update to show data in contexted of the selected split(s). In addition to the standard Dynatrace dashlets of *Transaction Flow*, *Response Time*, *Database*, *Errors* and *Web Requests*, we've added the following:

- *Portlet-based* Business Transaction splits
- *StrutsPortlet-based* Business Transaction splits.

These Business Transaction sets will show, for the selected CMS page, the portlet requests (*Portlet-based* splits) and *StrutsPortlet* portlet requests (*StrutsPortlet-based* splits) executed on given page, along with the average time it takes to complete the portlet execution portion of that PurePath. However, due to the nature of the Business Transaction limitations, the average time will be for all portlets of that type of Business Transaction on the page. For example:

- */web/guest/blogs*
    - The *Portlet Rendered* Business Transaction measures the amount of time it takes to execute the Liferay method `PortletContainerUtil.render()` and all of its children.
    - The Business Transaction further takes and splits the occurrence of `PortletContainerUtil.render()` by the *portletId*. In the PurePath, you'll see every instance and time for each *portletId* rendered (or action processed or resource served), however due to the nature of Business Transactions, the average time in `render()` is the average of all occurrences of `render()` in one PurePath, not the time for rendering *portletId* 23 vs. *portletId* 42.
    - Therefore, in the *Portlet Rendered* tab of this dashboard, each *portletId* will most likely display the same average time for one page, regardless of the *portletId*. By drilling down to the PurePaths, you can view the individual instances of rendering time for each portletId.

Please note, that the *Failure* percentage and status light will refer to the health of the page, not the portlet.

Liferay HTTP API Request Analysis

This dashboard has the same parent/child relationship as described in the Liferay CMS Page Performance Analysis dashboard, but since portlets are not part of these pages, the portlet tabs are not present.

Select any API request split to determine where time is being spent in each page request type. Drill down to PurePaths for a granular view.

Liferay Operations Overview

This dashboard provides an operations oriented high level view of the health of the Liferay hosts, processes and application performance.

Incident and Threshold alerts for *Host*, *Process* and *Application* metrics and placed on the top left side of the dashboard.

- Incident lights cover multiple out of the box incidents for each tier. If more incidents are created, they can be added.
- Threshold alert lights will turn yellow or red when a threshold is surpassed. Please note that the thresholds for each metric have to be added. You can edit these values by right clicking the light, select *Edit Measure*, or, if not available, configure series and then edit measure on the series.

Graphs present key host and process metrics.

Application Functional Health provides an overview of transaction count vs. health (pass/fail percent). It also includes error hotspots.

Transaction Duration With/Without GC is a 100% bar graph plotting the duration of a transaction vs. the duration of a transaction subtracting the time spent in garbage collection. The measure shows the direct impact of GC on transaction response time. Optimally, the line will ride on the 50% line, meaning GC is not impacting transaction performance. As Pure Path Duration (Red) creeps higher on the chart, this means GC is impacting transaction performance and GC activity should be investigated using Dynatrace's memory and GS analysis tools.

Liferay Ehcache Metrics

This dashboard provides an overview of Ehcache performance in Liferay cache system.

Cache Occupancy Rate (in-memory) is a calculated percentage of measure *MemoryStoredObjectCount* over *MaxElementsInMemory*. It tells how much space is occupied / left in cache of given type and whether particular cache's size needs adjustments.

The upper thresholds are set to 90% (warning) and 95% (error). If one or more caches are nearly or completely full (measure value is close to 100%), it might indicate the cache is too small for the size of Liferay data of given type. As a result, the size of that cache might need adjusting - making the cache bigger so that it can retain more frequently used elements.

In-Memory Hit Percentage reports the success rate of objects being found in cache when requested by Liferay. Under optimal conditions, the numbers should be as close to 100% as possible.

Liferay Process Monitoring

This dashboard can be used as an alternate to the Dynatrace out of the box process dashboards. The advantage of this one is that it rolls performance of all Java processes into one dashboard as opposed to the out of the box single process views. Each graph will represent data from all instrumented JVMs.

Process incident alerts, along with host and application transaction alerts show the status of the systems.

Roll over a datapoint in the graphs to get more detail on the chart readings. Alternately, double click the graph title to make that graph full screen for a more expanded view. Double click the title again to revert to the normal view.

Liferay Database Performance

This dashboard provides an overview of database performance from the Liferay's point of view.

Database Hotspots shows the queries being made in order of execution count. Selecting a query will show a tree below to trace which components are making the calls.

Database Connection Time vs. Connection Count plots the time it takes for the application to acquire a database connection vs. the number of connections.

Connection Time violation is pre-configured to warn when a connection takes more than 8ms and go severe on 10ms. Right click and edit measure to change thresholds.

Database Time vs. Count plots the time spent in database operations vs. the number of database executions. This includes connection and preparation time.

Slow SQL Statements shows the top 20 slowest SQL statements sorted by max time. Right click to drill down for more analysis.

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. Lifeary Developer Network: https://dev.liferay.com
4. Liferay 6.2 User Guide: https://www.liferay.com/documentation/liferay-portal/6.2/user-guide
5. Liferay 6.2 Developer's Guide: https://www.liferay.com/documentation/liferay-portal/6.2/development
6. Tomcat 7.0 Connectors:
    - https://tomcat.apache.org/tomcat-7.0-doc/connectors.html
    - https://tomcat.apache.org/tomcat-7.0-doc/config/ajp.html
    - https://tomcat.apache.org/tomcat-7.0-doc/config/http.html