eXo Platform 3.0 Administrators Guide

eXo Platform ()

Chapter 1. Introduction

1.1. Welcome to eXo Platform

eXo Platform 3.0, the user experience platform for Java, is comprised of Core and Extended Services.

- **Core Services**
  - *GateIn Portal*: a powerful framework for developing portlets and other web-based user interfaces
  - *eXo Content*: extends portal-based applications with Enterprise Content Management (ECM) capabilities
    - *eXo WCM*: web content management services
    - *xCMIS*: an implementation of the full stack of Java-based CMIS (Content Management Interoperability Specification) services on top of eXo WCM
    - *eXo Workflow*: integrated BPM (business process management) capabilities
  - *eXo IDE*: an intuitive web-based development environment that allows developers to build, test and deploy client applications (such as gadgets and mashups) and REST-ful services online
  - *CRaSH*: enables easy browsing of JCR trees, and serves as a shell for executing JCR operations
- **Extended Services**
  - *eXo Social*: a framework for building gadgets that can display and mash-up activity information for contacts, social networks, applications and services
  - *eXo Collaboration*: easily add Mail, Chat, Calendar and Address Book services to portal-based web applications
  - *eXo Knowledge*: adds Forum, Answers and FAQ functionality to portal-based apps, for collecting, organizing and sharing user knowledge

eXo Platform is a fully supported and commercially licensed product based on eXo open source projects. Designed for enterprise use, it has been packaged and tested to optimize production readiness and administration. eXo Platform runs on JBoss, Spring, Tomcat, WebSphere, and other Java applications servers, and can be used with most relational database systems, including MySQL and Oracle.

1.2. Who should read this guide?

This guide describes how to get started with eXo Platform, specifically for:

- *System Administrators* who want to use, deploy and manage eXo Platform system in their enterprise.
- *Developers* who want to know how to leverage eXo Platform in their customer projects.

This document will guide you through the most important tasks for eXo Platform administration and management. At the end, you will be able to install, configure, migrate, and manage your eXo Platform system.

Chapter 2. Installation

eXo Platform is packaged as a deployable enterprise archive, per the Java EE specification, and as a configuration directory.

2.1. Installing the Tomcat bundle

The easiest way to install eXo Platform is to take the default bundle. This is a ready-made package on top of Tomcat 6 application server, so you simply need to copy the `bin/tomcat6-bundle/` directory to your server.

2.2. Startup

eXo Platform leverages the Application Server on which it is deployed. This means, to start and stop eXo Platform, you only need to start and stop your application with the default commands.

- On Linux/Unix:

```
$TOMCAT_HOME/bin/gatein.sh
```

- On Windows:

```
%TOMCAT_HOME%\bin\gatein.bat
```

The server has started when you see the following message in your log/console:

```
INFO: Server startup in 353590 ms
```

2.3. Shutdown

- On Linux/Unix:

```
$TOMCAT_HOME/bin/shutdown.sh
```

- On Windows:

```
%TOMCAT_HOME%\bin\shutdown.bat
```

The server has stopped when you see the following message in your log/console:

```
INFO: Stopping Coyote HTTP/1.1 on http-8080
```

2.4. Startup scripts

eXo comes with several builtin startup scripts :

- gatein.sh : start eXo on Linux / Mac
- gatein.bat : start eXo on Windows
- gatein-dev.sh : start eXo on Linux / Mac in developer mode
- gatein-dev.bat : start eXo on Windows in developer mode

2.4.1. Normal Mode

The normal mode starts with the following JVM options :

```
-Xms256m
-Xmx1024m
-XX:MaxPermSize=256m
-Djava.security.auth.login.config=../conf/jaas.conf
-Dexo.conf.dir.name=gatein/conf
-Dexo.profiles=default
```

`-Xms`	Minimal heap size (defaults to 256 MB)
`-Xmx`	Maximal Heap Size of (defaults to 1 GB)
`-Djava.security.auth.login.config`	path to the JAAS security file where the security domains are and JAAS authentication modules are declared
`-Dexo.conf.dir.name`	path where eXo will start looking at configuration.properties and configuration.xml
`-Dexo.profiles`	the list of comma-separated exo profiles to activate

This is enough to start and run a demo, but you will need to adjust these values for a production setup.

2.4.2. Developer mode

The developer mode scripts add a few more options.

```
-Xdebug -Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n
-Dcom.sun.management.jmxremote
-Dorg.exoplatform.container.configuration.debug
-Dexo.product.developing=true
```

`-Dcom.sun.management.jmxremote`	activates the JMX remoting
~~`-Xdebug Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n`~~	enables remote debugging
`-Dorg.exoplatform.container.configuration.debug`	the container will log to the console what xml files it loads
`-Dexo.product.developing=true`	desactivates javascript and css merging for easier

	debugging

2.5. eXo Profiles

By passing `-Dexo.profiles=p1,p2,p3...`, you can enable modules.

You can enable/disable select modules if they are not in use.

- `collaboration`: enables eXo Collaboration module
- `knowledge`: enables eXo Knowledge module
- `social`: enables eXo Social module
- `workflow`: enables the Workflow add-on within the eXo Content module

Additionally, you can use several predefined profiles:

- (none): contains GateIn + WCM
- `default`: contains all except workflow (gatein,ide,wcm,collaboration,social,knowledge)
- `all`: all available modules

Note

Profiles are pluggable, so you can combine them together to shape eXo Platform to your needs. e.g : `-Dexo.profiles=default,workflow` will take the default profile and workflow

Chapter 3. Configuration

3.1. eXo Configuration

In eXo platform, the configuration lives in a folder whose location is controlled by a system property named `exo.conf.dir`. By default, the `gatein.sh` startup script sets this property as follows :

```
-Dexo.conf.dir.name=gatein/conf
```

So the main entry point for the eXo Platform's configuration is `/gatein/conf/`. This directory contains the following files :

- `configuration.properties` : the main **system configuration**
- `configuration.xml` : contains the default **portal container** configuration
- `portal/portal/configuration.xml` : the main **external customization** entry point for the default portal container.

3.1.1. Portal Containers, Customization and Configurations

Let's stop and explain some parts of the eXo internals in order to understand the roles of these configuration files.

eXo Platform kernel groups runtime components in *portal containers*. A portal container holds all components to run a portal instance. It serves portal pages under the servlet context for his name.

The default portal container in eXo Platform is called simply "portal". This is why the default url for the samples is http://localhost:800/*portal*.

The default portal container can be configured directly inside `exo.conf.dir`.

But eXo Platform is capable of running several portal instances simultaneously on the same server. Each instance can be configured and customized independently via files located at : `/gatein/conf/portal/$PORTAL_NAME`, where $PORTAL_NAME is the name of the portal container.

> **Note**
>
> The exact name of the configuration file can be altered. Please refer to the section dedicated to *PortalContainerDefinition* in the Kernel reference for more details on portal containers and other options to modify the location of the properties.

Services that run inside a portal container are declared via xml configuration files like configuration.xml. Such files exists in jars, wars and below `exo.conf.dir`.

XML configuration files also serve as the main way to customize the portal via the multiple plugins offered by eXo components.

Additionally, xml files may contain variables that are populated via properties defined in

configuration.properties. Hence, the configuration.properties serves as exposing some chosen variables that are necessary to configure eXo Platform in a server environment.

3.1.2. configuration.properties

The system configuration is done mostly in `configuration.properties`. In most cases, this should be the only file a system administrator will need to configure.

In the Tomcat bundle, this file is located at `/gatein/conf/configuration.properties`

3.1.3. configuration.xml

This file contains the builtin configuration for the "portal" portal container.

In most cases, you should not need to change this file.

In case your project does not want to use "portal" as the default portal, this file can be used to to import another *PortalContainerDefinition* into the root container.

> **Note**
>
> Details on how to configure a new portal container are out of the scope of this book but is extensively covered in the kernel reference guide.

3.1.4. portal/portal/configuration.xml

This file is empty by default. This is where further customizations can be placed. Generally, custom configurations are provided by extension wars. But this file is the last loaded by the kernel. It has a higher priority over any other configuration file, including extensions. So it let's you override any internal component configuration.

This may turn very handy for services or configurations that are not exposed in configuration.properties, but you'd like to tune anyway.

For example, you could decide to change the default transaction timeout to 2 minutes with this piece of xml:

```
<component>
    <key>org.exoplatform.services.transaction.TransactionService</key>
    <type>org.exoplatform.services.transaction.jbosscache.JBossTransactionsService</type>
    <init-params>
      <value-param>
        <name>timeout</name>
        <value>120</value>
      </value-param>
    </init-params>
  </component>
```

3.2. Database Configuration

eXo Platform relies on the application server for its database access, so the database must be configured as a datasource at the AS level. That datasource is obtained by accessing the enterprise naming context (ENC) through the Java Naming and Directory Interface (JNDI) service.

By default, eXo Platform defines two datasources:

- `exo-jcr` - for the Java Content Repository (JCR).
- `exo-idm` - for the organizational model.

The tomcat bundle comes with the two datasources preconfigured as GlobalNamingContext Resources. Please refer to Tomcat's JNDI Resources How To for more details on JNDI resources binding in Tomcat.

The configuration lives in 3 files that you will want to edit in order to change the database.

3.2.1.1. configuration.properties

```
$TOMCAT_HOME/gatein/conf/configuration.properties
```

Indicate to eXo the name of the datasources.

```
# JNDI name of the datasource that will be used by eXo JCR
gatein.jcr.datasource.name=java:/comp/env/exojcr
...
# JNDI Name of the IDM datasource
gatein.idm.datasource.name=java:/comp/env/exo-idm
```

eXo will automatically append the portal container name ("*portal" by default) to these values before it performs a JNDI lookup.*

3.2.1.2. server.xml

```
$TOMCAT_HOME/conf/server.xml
```

Declare the binding of the datasources in the GlobalNaming context :

```
<!-- eXo JCR Datasource for portal -->
<Resource name="exo-jcr_portal" auth="Container" type="javax.sql.DataSource"
               maxActive="20" maxIdle="10" maxWait="10000"
               username="sa" password="" driverClassName="org.hsqldb.jdbcDriver"
               url="jdbc:hsqldb:file:../gatein/data/hsql/exo-jcr_portal"/>

<!-- eXo IDM Datasource for portal -->
<Resource name="jdbc/exo-idm_portal" auth="Container" type="javax.sql.DataSource"
               maxActive="20" maxIdle="10" maxWait="10000"
               username="sa" password="" driverClassName="org.hsqldb.jdbcDriver"
               url="jdbc:hsqldb:file:../gatein/data/hsql/exo-idm_portal"/>
```

3.2.1.3. starter.xml

```
$TOMCAT_HOME/conf/Catalina/localhost/starter.xml
```

We declare resource links that make our datasources accessible to the starter webapp which is used when starting eXo.

```
<ResourceLink global="exo-jcr_portal" name="exo-jcr_portal" type="javax.sql.DataSource"/>

<ResourceLink global="exo-idm_portal" name="exo-idm_portal" type="javax.sql.DataSource"/>
```

3.3. FileSystem pathes

eXo needs read/write access to several pathes in the local filesystem.

```
gatein.data.dir=../gatein/data

# path for any JCR data
gatein.jcr.data.dir=${gatein.data.dir}/jcr

# path for file data inserted in JCR
gatein.jcr.storage.data.dir=${gatein.jcr.data.dir}/values

# path for the jcr index
gatein.jcr.index.data.dir=${gatein.jcr.data.dir}/index
```

The following table explains what goes in what path. The `temporary?` column indicates if the data is temporary or persistent.

variable	content	temporary
`gatein.data.dir`	jta transactional data	yes
`gatein.jcr.data.dir`	jcr swap data	yes
`gatein.jcr.storage.data.dir`	binary value storage for jcr	no
`gatein.jcr.index.data.dir`	lucene index for JCR	no

Each variable can be defined as an absolute or relative path. The default configuration combines them to obtain a compact tree :

```
/gatein       # gatein.data.dir
  /data
    /hsql
    /jcr      # gatein.jcr.data.dir
      /index  # gatein.jcr.index.data.dir
      /swap
      /values # gatein.jcr.storage.data.dir
    /jta
```

3.4. Mail Server

eXo Platform requires an SMTP server in order to send emails such as notificaitons or password reminders.

```
gatein.email.smtp.username=
gatein.email.smtp.password=
gatein.email.smtp.host=smtp.gmail.com
gatein.email.smtp.port=465
gatein.email.smtp.starttls.enable=true
gatein.email.smtp.auth=true
gatein.email.smtp.socketFactory.port=465
```

```
gatein.email.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
```

More details con be found in the JavaMail API documentation.

3.5. WebDAV Cache Control

The embedded WebDAV server lets you control the cache-control http header that it transmits to clients by mimetype. This is useful for fine-tuning your website.

The configuration property is: `exo.webdav.cache-control`

```
exo.webdav.cache-control=text/*:max-age=3600;image/*:max-age=1800;*/*:no-cache;
```

The property expects a comma-separated list of key=pair values, where keys are a list of mimetypes followed by the cache-control value to set.

3.6. Chat Server

3.6.1. XMPPMessenger

If you changed the hostname and port for the chat server, then you'll need to edit two properties:

```
# IP or hostname for the eXo Chat XMPP server
exo.chat.server=127.0.0.1

# TCP port for where the eXo Chat server listens for XMPP calls
exo.chat.port=5222
```

3.6.2. Chat server configuration

The standalone Chat server is configured in the file `$CHATSERVER/conf/openfire.xml`.

Configuration is based on properties expressed in an XML syntax. For example, to set property `prop.name.is.blah=value`, you would write this xml snippet :

```
<prop><name><is><blah>value</blah></is></name></prop>
```

Openfire has an extensive list of configuration properties. Please read the list of all properties in Openfire documentation

The chat server is an openfire server bundled with plugins and configurations that allow connectivity to eXo Platform. The following properties are used to configure it.

Property	Description	Default value
env		
serverbaseURL	base url for all URLs below	http://localhost:8080/
restContextName	name of the rest context	rest
provider		
authorizedUser.name	username to authenticate against the HTTP REST service	root
authorizedUser.password	password matching with provider.authorizeduser.name	password
eXoAuthProvider		
authenticationURL	URL to authenticate users	/organization/authenticate/
authenticationMethod	HTTP method used to pass parameters	POST
eXoUserProvider		
findUsersURL	URL to find all users	/organization/xml/user/find-all/
findUsersMethod	HTTP method for user/find-all	GET
getUsersURL	URL to retrieve a range of users	/organization/xml/user/view-range/
getUsersMethod	HTTP method for user/view-range	GET
usersCountURL	URL to count users	/organization/xml/user/count/
usersCountMethod	HTTP method for user/count	GET
userInfoURL	URL to get user info	/organization/xml/user/info/
userInfoMethod	HTTP method for user/info	GET
eXoGroupProvider		
groupInfoURL	URL to get group info	/organization/xml/group/info/
groupInfoMethod	HTTP method for info	GET
getGroupsAllURL	URL to view all groups	/organization/xml/group/view-all/
getGroupsAllMethod	HTTP method for group/view-all	GET
getGroupsRangeURL	URL to view a group range	/organization/xml/group/view-from-to/
getGroupsRangeMethod	HTTP method for group/view-from-to	GET
getGroupsForUserURL	URL to get groups for a user	/organization/xml/group/groups-for-user/
getGroupsForUserMethod	HTTP method for groups-for-user	GET
groupsCountURL	URL to count groups	organization/xml/group/count
groupsCountMethod	HTTP method for group/count	GET

3.6.2.1. Ports

In order to run properly the chat server needs several ports to be opened in the firewall.

Port	Type	Description
5222 (1)	client to server (xmpp)	The standard port for clients is to connect to the server. Connections may or may not be encrypted. You can update the security settings for this port with `exo.chat.port` property.
9090 && 9091	Admin Console (http)	The port used for respectively the unsecured and secured Openfire Admin Console access.
3478 & 3479	STUN service	The port used for the service that ensures connectivity between entities when behind a NAT.

3.7. Logging

Logging in eXo Platform is controlled by the Java Logging API.

By default, logging is configured to:

- log errors and warnings on the console
- log info level statements in **/gatein/logs/gatein-YYYY-MM-DD.log**

In Tomcat, the logging is configured via the conf/logging.properties file. Please refer to Tomcat's Logging Documentation for more information on how to adjust this file to your needs.

3.8. JCR

A set of properties control the behaviour of the JCR.

```
# Type of JCR configuration to use. Possible values are :
# local : local JBC configuration
# cluster : cluster JBC configuration
gatein.jcr.config.type=local

# This is the filter used to notify changes in the jcr index
# in cluster mode, use org.exoplatform.services.jcr.impl.core.query.jbosscache.JBossCacheIndexChangesFilter
gatein.jcr.index.changefilterclass=org.exoplatform.services.jcr.impl.core.query.DefaultChangesFilter

# JCR cache configuration
gatein.jcr.cache.config=classpath:/conf/jcr/jbosscache/${gatein.jcr.config.type}/config.xml

# JCR Locks configuration
gatein.jcr.lock.cache.config=classpath:/conf/jcr/jbosscache/${gatein.jcr.config.type}/lock-config.xml

# JCR Index configuration
gatein.jcr.index.cache.config=classpath:/conf/jcr/jbosscache/cluster/indexer-config.xml
```

```
gatein.jcr.jgroups.config=classpath:/conf/jcr/jbosscache/cluster/udp-mux.xml
```

gatein.jcr.config.type	**use `cluster` is you want to use eXo Platform in cluster mode. Otherwise leave `local`**
gatein.jcr.index.changefilterclass	**in cluster mode change it to `org.exoplatform.services.jcr.impl.core.query.jbosscache.` **
gatein.jcr.cache.config	JBoss Cache configuration for the JCR locks
gatein.jcr.index.cache.config	JBoss Cache Configuraiton for the JCR index
gatein.jcr.jgroups.config	JGroups configuration to use for cluster mode

Please refer to the JCR reference guide for the details of configuring these files.

Chapter 4. Management

4.1. Management of eXo Platform

Management of the resources of a platform is critical for production usage. The eXo Platform product is exposed as a manageable set of resources that can be inspected at runtime to monitor and manage servers.

When it comes to Java, the Java Management Extension (also known as JMX) is the de-facto standard to expose managed resources externally.

This chapter explains the various resources provided by the eXo Platform server, the management actions that can be performed, and how to obtain relevant metrics.

4.1.1. JMX Interface

Resource management is exposed via the JMX layer. eXo Platform registers a set of MBean entities in an MBeanServer.

At runtime, the MBeans are registered by the eXo Kernel in the MBeanServer, which the application server creates, and the MBeans are directly viewable in the JMX console. However, we do advocate using a better JMX client such as the JVisualVM, available since Java 6.

In order to enable JMX monitoring in Tomcat, you need to pass the following system property to the VM : `-Dcom.sun.management.jmxremote`

4.1.2. REST Interface

The built-in REST Managment Provider of GateIn makes some of the MBeans operations accessible as REST endpoints. Administrators could handle the system simply with a browser; no complex configuration needs to be performed.

Only members of the platform/administrators group are given permission to work on REST management. The authentication requires login/password of the user's account.

The base URL to access the REST endpoints is http://localhost:8080/rest/managment, with the last one followed by the parameter parsed in the managed resource's @RESTEndpoint annotation, leading slash then targeted operation. Consider the SkinService, which is annotated @RESTEndpoint("skinservice"); the full URL to access JMX 'getSkinList' method through REST request is http://localhost:8080/rest/management/skinservice/getSkinList

4.2. Management view

4.2.1. Kernel management view

4.2.1.1. PortalContainer

PortalContainer manages all objects and configurations for a given portal. The JMX name is `exo:container=portal,name="portal"`

- `configurationXML`: provides the effective runtime configuration calculated by the loading mechanism. URLs used to load parts are indicated to help.
- `registeredComponentNames`: provides the list of all components registered.

4.2.2. Cache management view

Cache is essential for improving the performance of a server infrastructure.

4.2.2.1. Cache management and statistics

Each cache is exposed and provides statistics and management operations. The JMX name of each cache MBean uses the template `exo:service=cache,name=CacheName` where CacheName is the name of the cache.

Attribute	Description
`Capacity`	the maximum capacity of the cache
`HitCount`	the number of times the cache was successfully queried
`MissCount`	the number of times the cache was queried without success
`Size`	gives the number of entries in the cache
`TimeToLive`	attribute gives the maximum lifetime that an entry is considered valid. A value of `1` means that the entries will never become stale

Operation	Description
`clearCache()`	evicts all the entries from the cache. It can be used to force a programmatic flush of the cache

4.2.2.2. Cache service

The cache service manages the different caches. The JMX name is `exo:service=cachemanager`.

Operation	Description
`clearCaches()`	force a programmatic flush of all the registered caches

4.2.2.3. PicketLink Cache

Picketink is the default implementation for the organization model. It uses its own cache JMX management interface under the name `exo:portal="portal",service=PicketLinkIDMCacheService,name=plidmcache`. `invalidateAll()`: invalidate all caches `invalidate(namespace)`: invalidates a specific cache namespace

4.2.3. Content management view

WCM provides a management view for:

4.2.3.1. WCM Composer

The WCM Composer is responsible for assembling the pages, and is key for serving pages efficiently. The JMX name is `exo:portal="portal",service=composer,view=portal,type=content`

- `Cached`: whether the cache is used or not
- `CachedEntries`: how many nodes are in the cache
- `cleanTemplates()`: cleans all templates in the cache
- `setCached(cached)`: enables/disables caching in the composer

4.2.4. Portlet Container management view

Portlet containers do provide a management view that consists mainly of statistic exposure. The JMX name of a portlet container follows this template: `exo:application="Application",portlet="Portlet"` where `Application` is the name of web application and `Portlet` is the portlet name in the web application.

4.2.4.1. Portlet Container statistics

Each portlet container has a corresponding managed resource that exposes statistics as attributes. The Portlet lifecycle has several phases (named action, event, render and resource). These are particular operations that a portlet can execute, and each is monitored.

Attribute	Description
`ActionCount`, `EventCount`, `RenderCount` and `ResourceCount`	the number of executions of a particular portlet phase
`MeanActionTime`, `MeanEventTime`, `MeanRenderCount` and `MeanResourceCount`	the average time spent during the execution of a particular portlet phase
`MaxActionTime`, `MaxventTime`, `MaxRenderCount` and `MaxResourceCount`	the maximum time spent during the execution of a particular portlet phase
`MinActionTime`, `MinEventTime`, `MinRenderCount` and `MinResourceCount`	the minimum time spent during the execution of a particular portlet phase

4.2.5. Portal management view

The portal management exposes the portal resources either for management or for obtaining statistics.

4.2.5.1. Template statistics

The template management exposes the various templates used by the portal and its components to render markup. Various statistics are available for individual templates, as well as aggregated statistics, such as the list of the slowest templates. Most of the management operations are performed on a single template; those operations take the template identifier as an argument. The JMX name of the template statistics MBean follows the template: `exo:view=portal,service=statistic,type=template`.

Attribute	Description	
`TemplateList`	returns the list of the loaded templates	
`SlowestTemplates`	returns the list of the 10 slowest templates	
`MostExecutedTemplates`	returns the list of the 10 most used templates	
`FastestTemplates`	returns the list of the 10 fastest templates	

Operation	Description
`getAverageTime(templateId)`	returns the average time spent in the specified template
`getExecutionCount(templateId)`	returns the number of times the specified template has been executed
`getMinTime(templateId)`	returns the minimum time spent in the specified template
`getMaxTime(templateId)`	returns the maximum time spent in the specified template

4.2.5.2. Template management

Template management provides the capability to force the reload of a specified template. The JMX name of the template management MBean follows the template `exo:view=portal,service=management,type=template`.

Operation	Description
`reload(templateId)`	reloads a specified template. The `reload()` operation reloads all the templates

4.2.5.3. Skin management

Attribute	Description
`SkinList`	the list of the loaded skins by the skin service

Operation	Description
`reloadAll(skinId)`	forces a reload of the specified skin and the operation `reloadSkins()` forces a reload of the loaded skins

4.2.5.4. Token Store

Caution

TODO

GateIn uses an internal store for tokens.

```
exo:portal="portal",service=TokenStore,name="memory-token"
exo:portal="portal",service=TokenStore,name="jcr-token"
exo:portal="portal",service=TokenStore,name="gadget-token"
```

4.2.5.5. Portal statistics

The JMX name of the portal statistics MBean follows the template `exo:view=portal,service=statistic,type=portal`.

Attribute	Description
`PortalList`	returns the list of the loaded portals

Operation	Description
`getThroughput(portalId)`	returns the number of request per second for the specified portal
`getAverageTime(portalId)`	returns the average time spent in a specified portal
`getExecutionCount(portalId)`	returns the number of times a specified portal has been executed
`getMinTime(portalId)`	returns the minimum time spent in the specified portal
`getMaxTime(portalId)`	returns the maximum time spent in the specified portal

4.2.5.6. Application statistics

Various applications are exposed to provide relevant statistics. The JMX name of the application statistics MBean follows the template: `exo:view=portal,service=statistics,type=application`.

Attribute	Description
`ApplicationList`	returns the list of the loaded applications
`SlowestApplications`	returns the list of the 10 slowest applications
`MostExecutedApplications`	returns the list of the 10 most executed applications
`FastestApplications`	returns the list of the 10 fastest applications

Operation	Description
`getAverageTime(applicationId)`	returns the average time spent in the specified

Operation	Description
	application
`getExecutionCount(applicationId)`	returns the number of times the specified application has been executed
`getMinTime(applicationId)`	returns the minimum time spent in the specified application
`getMaxTime(applicationId)`	returns the maximum time spent in the specified application

4.2.6. Knowledge Management view

Note

TODO

4.2.6.1. BBCodes

Note

TODO

4.2.6.2. Forum Service

Note

TODO

4.2.6.3. Jobs

Note

TODO

4.2.6.4. Plugins

Note

TODO

4.2.6.5. Storage

Note

TODO

Chapter 5. Clustering

5.1. When should you consider clustering ?

Installing eXo platform in cluster mode should be considered in the following cases:

- Load Balancing : when a single single server node is not enough to handle the load
- High Availability : when you want to avoid a single point of failure by having redundant nodes

These characteristics should be handled by the overall architecture of your system. Load Balancing is typically achieved by a front server or device that distributes the request to the cluster nodes. Also, high availability on the data layer can be typically achieved using the native replication implemented by RDBMS.

In this chapter, we will cover only the changes needed by eXo to work in a cluster.

5.2. Shared file system

In eXo Platform, the persistence mostly relies on JCR, which is a middleware between the eXo applications (including the portal) and the database. Hence this component must be configured to work in cluster.

The embedded JCR server requires a portion of its state to be shared on a file system shared among cluster nodes :

- the values storage
- the index

We strongly advise the use of a mount point on a SAN.

5.3. Configuration setup

The switch to a cluster configuration is done in `configuration.properties`

First, switch the JCR to cluster mode:

```
gatein.jcr.config.type=cluster
gatein.jcr.index.changefilterclass=org.exoplatform.services.jcr.impl.core.query.jbosscache.JBossCacheIndexChange
```

This will tell the JCR to enable automatic network replication and discovery between other cluster nodes.

Next, configure the path for the shared filesystem :

```
gatein.jcr.storage.data.dir=/PATH/TO/SHARED/FS/values
gatein.jcr.index.data.dir=/PATH/TO/SHARED/FS/index
```

The path is shared, so all nodes will need read/write access to this path.

5.3.1. advanced configuration

The cluster mode is preconfigured to work out of the box. It relies on the JBoss Cache configuration.

```
# JCR cache configuration
gatein.jcr.cache.config=classpath:/conf/jcr/jbosscache/${gatein.jcr.config.type}/config.xml

# JCR Locks configuration
gatein.jcr.lock.cache.config=classpath:/conf/jcr/jbosscache/${gatein.jcr.config.type}/lock-config.xml

# JCR Index configuration
gatein.jcr.index.cache.config=classpath:/conf/jcr/jbosscache/cluster/indexer-config.xml
gatein.jcr.jgroups.config=classpath:/conf/jcr/jbosscache/cluster/udp-mux.xml
```

Note

TODO : find and indicate the location of these files in jars. Give a link to the JGroups / JBoss Cache relevant doc

5.4. Startup option

You need to indicate the *cluster* kernel profile to eXo Platform. This can be done by editing `gatein.sh` like this:

```
EXO_PROFILES="-Dexo.profiles=default,cluster"
```

5.5. Cluster management

5.5.1. Cluster initialization

For the very first startup of your JCR cluster, you should only start a single node. This node will initialise the internal JCR database and create the system workspace. Once this first node is deinitely started, you can start the other nodes.

Note

This contraint is only for the very first start. Once the initialization has been done, you can start odes in any order

5.5.2. Starting and Stoping

Nodes of the cluster will automatically try to join others at startup. Once they discover each other, they will synchronize their state. During the synchronization the node is not ready to serve requests.

Note

How does an admin validate that teh cluster works properly ? Is there something noticeable on screen ?

5.5.3. From local to cluster mode

If you intend to migrate your production system from local (non cluster) mode to cluster, follow these steps :

- Update the configuration to cluster mode as explained above on your main server
- Use the same configuration on other cluster nodes
- Move the index and value storage to the shared file system
- Start the cluster

Chapter 6. Deployment

6.1. How to remove the sample applications

eXo Platform comes with two sample portals that showcase the capabilities of the product. Before deploying your system in production, you will want to remove these sample apps.

> **Caution**
>
> The instructions below assume that you are using the hsqldb embedded database configuration.

6.1.1. Removing Acme Website

1. Stop the server `shutdown.sh`
2. Delete acme-portal.war
3. Delete exo.ecms.ext.acme.config.jar
4. Delete `gatein/data`
5. Restart

6.1.2. Removing Acme Social Intranet

1. Stop the server `shutdown.sh`
2. Delete office-portal.war
3. Delete exo.platform.office.config.jar
4. Delete `gatein/data`
5. Restart

6.2. How to deploy a custom extension

Extensions are packaged as java EE web applications and come packaged as normal .war files. Hence, to deploy a custom extension, you will likely do as for any other webapp. In Tomcat this ends up by copying the war archive to the `webapps` folder

However, note that the Gatein extension mechanism imposes that the starter.war webapp starts after all extension wars. This is the case for the sample applications bundled by defaut, but you must ensure that for your custom applications. There are several ways to control le loading order of webapps in Tomcat. Please refer to Tomcat's Deployer How To

6.3. How to setup an Apache Frontend

It may be necessary to use an HTTP server as a frontend for tomcat. For example, you may want to keep more then one application server on the same host, and/or you want to access these app servers with separate DNS names, without having to add a port to the URL. There are two methods that allow you to "*glue*" Apache HTTP Daemon and tomcat application server:

- via HTTP protocol, using proxy module
- via Apache JServ Protocol, using tomcat connector or HTTPD AJP proxy module

6.3.1. Base configuration for apache

First, you need to configure a new virtual host in Apache HTTPD for the application server. This is the simplest example of a virtual host:

```
<VirtualHost *:80>
        ServerName      Enter your server DNS name here
        RedirectMatch permanent "^/?$" "/portal/"
</VirtualHost>
```

You can find more information about Apache HTTP daemon host here

6.3.2. Connection via HTTP protocol

With the *glue* method, it is necessary to configure Apache HTTP daemon to work as **reverse** proxy, which will redirect the client's requests to the app server's HTTP connector. For this type of connection, you will need to include the **mod_proxy** module in the HTTP demon configuratinon file. This can be found in the **httpd.conf** file, which is usually located here: **/etc/httpd/conf/**. However, depending on your OS, this path may vary. You will then need to add some directives to your virtual host configuration.

```
ProxyRequests   Off
ProxyPass  "/" http://YOUR_AS_HOST:AS_HTTP_PORT/
ProxyPassReverse "/" http://YOUR_AS_HOST:AS_HTTP_PORT/
```

Note

In the example above: YOUR*AS*HOST - host (IP or DNS name) is the location of your application server. If you run HTTP demon on the same host as your app server, you can change this to **localhost**. AS*HTTP*PORT - port, is the location where your app server will listen for incoming requests. For tomcat this value, by default, is 8080. You can find the value at **tomcat/conf/server.xml**

In this example, HTTP daemon will work in **reverse proxy** mode (ProxyRequests Off) and will redirect all requests to tcp port 8080 on localhost. So, the configuration of a virtual host will look like the following:

```
<VirtualHost *:80>
        ServerName      Enter your server DNS name here
        RedirectMatch permanent "^/?$" "/portal/"
        ProxyRequests   Off
        ProxyPass  "/" http://localhost:8080/
```

```
                ProxyPassReverse "/" http://localhost:8080/
        </VirtualHost>
```

For more detail about **mod_proxy**, review this documentation

6.3.3. Connection via AJP protocol

As described above, the 'glue' method can be implemented in two ways:

- using the native Apache HTTP demon's AJP proxy module
- using the native Apache Tomcat's AJP conector

With the first method, you only need the HTTP demon and application server, but settings are limited. With the second method, you can obtain much richer settings, but you will need to download and install additional modules for HTTP Daemon that are not included in the default package.

6.3.3.1. AJP proxy module

Make sure that **mod_proxy_ajp.so** is included in the list of loadable modules. Add the following to your virtual host configuration setting:

```
ProxyPass / ajp://localhost:8009/
```

In this example, the app server is located on the same host as the Apache HTTP daemon, and accepts incoming connections on port 8009 (the default setting for tomcat application server). A full list of virtual host configurations can be found here:

```
        <VirtualHost *:80>
                ServerName      Enter your server DNS name here
                RedirectMatch permanent "^/?$" "/portal/"
                ProxyRequests   Off
                ProxyPass / ajp://localhost:8009/
        </VirtualHost>
```

6.3.3.2. Apache Tomcat's AJP connector

1. Download AJP connector module from here
2. Move the downloaded **mod_jk.so** file into HTTPD's module directory. For example: **/etc/httpd/modules** (this may be different, depending on the OS)
3. Create the configuration file for module modjk.conf

```
        LoadModule    jk_module  modules/mod_jk.so
        <IfModule jk_module>
          # ---- Where to find workers.properties
          JkWorkersFile   conf.d/workers.properties
          # ---- Where to put jk logs
          JkLogFile       logs/mod_jk.log
          # ---- Set the jk log level [debug/error/info]
          JkLogLevel    info
          # ---- Select the timestamp log format
          JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
```

```
  JkRequestLogFormat "%w %R %T"
  # ---- Send everything for context /examples to worker named worker1 (ajp13)
  JkMountFileReload       "0"
</IfModule>
```

You can find more details in the Tomcat docs

1. Place the **mod_jk.conf** file into the directory where other configuration files for Apache HTTP Demon are located. For example, **/etc/httpd/conf.d/**

2. Create a workers.properties file, which defines AJP workers for HTTP demon.

```
worker.list=status, WORKER_NAME
# Main status worker
worker.stat.type=status
worker.stat.read_only=true
worker.stat.user=admin
# Your AJP worker configuration
worker.WORKER_NAME.type=ajp13
worker.WORKER_NAME.host=localhost
worker.WORKER_NAME.port=8109
worker.WORKER_NAME.socket_timeout=120
worker.WORKER_NAME.socket_keepalive=true
```

Note

In example above you can change *WORKER**NAME** to any value.*

1. Place this file in the same directory as the mod_jk.conf file.

2. Update the virtual host configuration:

```
<VirtualHost *:80>
        ServerName      Enter your server DNS name here
        RedirectMatch permanent "^/?$" "/portal/"
        ProxyRequests   Off
        JkMount         /* WORKER_NAME
</VirtualHost>
```

Chapter 7. Backup and Recovery

7.1. How to backup eXo

Note

TODO : explain how to backup an eXo instance. This involves backing up the databases and the filesystem for index and value storage