



Magnolia International Ltd.

## **Performance and high availability**

Magnolia CMS Tech Brief

Magnolia International Ltd.

10/13/2010

Copyright Magnolia International Ltd. 2010.

Magnolia is a registered trademark of Magnolia International Ltd.

All rights reserved. No part of this document may be reproduced in any form without prior permission in writing from Magnolia International Ltd.

## Table of contents

<b>Certified stack</b> .....	<b>4</b>
Summary .....	5
<b>Local database</b> .....	<b>6</b>
<b>Instances</b> .....	<b>7</b>
Instance configuration examples .....	9
<b>Content authoring and activation</b> .....	<b>10</b>
<b>Clustering</b> .....	<b>11</b>
Jackrabbit clustering .....	11
Author clustering .....	11
<b>Disaster recovery and backup</b> .....	<b>13</b>
<b>Caching</b> .....	<b>14</b>
Compression .....	14
Advanced cache strategies .....	15

By good performance we understand the capability of a system to serve multiple simultaneous requests in reasonable time under heavy load. In Magnolia CMS terms this means that editors can edit without delay, content is activated to public instances instantly, and public instances serve requests without perceivable slowness.

High availability refers to the degree of operational continuity in a system. A typical metric is the percentage of uptime relative to "100% operational". In a high availability system redundant instances ensure that services are available even when a instance goes down.

In this brief we examine how Magnolia ensures great performance and high availability.

## Certified stack

Certified stack is a collection of software components that have been tested and are guaranteed to work with Magnolia CMS and the Jackrabbit repository. A key criterion in stack selection is performance. Installing a Magnolia instance on top of a certified stack ensures optimal performance.

### Certified operating systems

- Linux distributions running kernel 2.6.
  - Ubuntu 8 and later
  - SuSE Linux Enterprise Server 10 and later
  - Fedora 8 and later
  - Red Hat Enterprise Linux Server 4 and later
  - CentOS 5 and later
  - Debian 5 and later
- Windows
  - Windows 2008 Server
  - Windows 2003 Server Standard Edition SP3
  - Windows XP Enterprise Edition SP2, Windows Vista, Windows 7
- Other
  - FreeBSD 7 or later
  - Mac OS X 10 or later

### Java

- Sun JVM 1.5.0 (build 14 and higher)
- Sun JVM 1.6.x

Sun JVM 1.5.0 build 14 and higher and Sun JVM 1.6.x are supported on all OSs and application servers.

### Application servers

- Websphere 6.1 SP2 or with higher service pack
- Oracle Weblogic Server version 9.2
- Oracle Weblogic Server version 10.1 and higher
- Tomcat 5.5.27 and higher
- JBoss 4.0.5 or higher

### Notes:

- Weblogic 9.2 does not support the Imaging module
- Weblogic and Websphere are supported only with the special Magnolia CMS package for Weblogic and Websphere
- Weblogic and Websphere are supported only with Magnolia Enterprise Edition Pro

### Databases

#### Embedded

- Derby 10.3.1.4 (the packaged version)

**External DB**

- MySQL 5.0 or higher with MySQL Connector 5.1.x (JDBC Drivers).
- Oracle 10g Enterprise Edition or higher

**Notes:**

- Derby DB 10.3.1.4 is supported only as an embedded DB, packaged version for low volume use only
- MySQL DB 5.0 or higher is supported as an external DB.
- InnoDB storage engine for MySQL is supported by Magnolia, MyISAM is not.
- Oracle 10g Enterprise Edition or higher is supported as an external DB

**Repositories**

- Jackrabbit (JR 1.0.1 for Magnolia CMS 3.0; JR 1.3.3 for Magnolia CMS 3.5; JR 1.4.5 for Magnolia CMS 3.6; JR 1.6 or higher for Magnolia CMS 4.x)

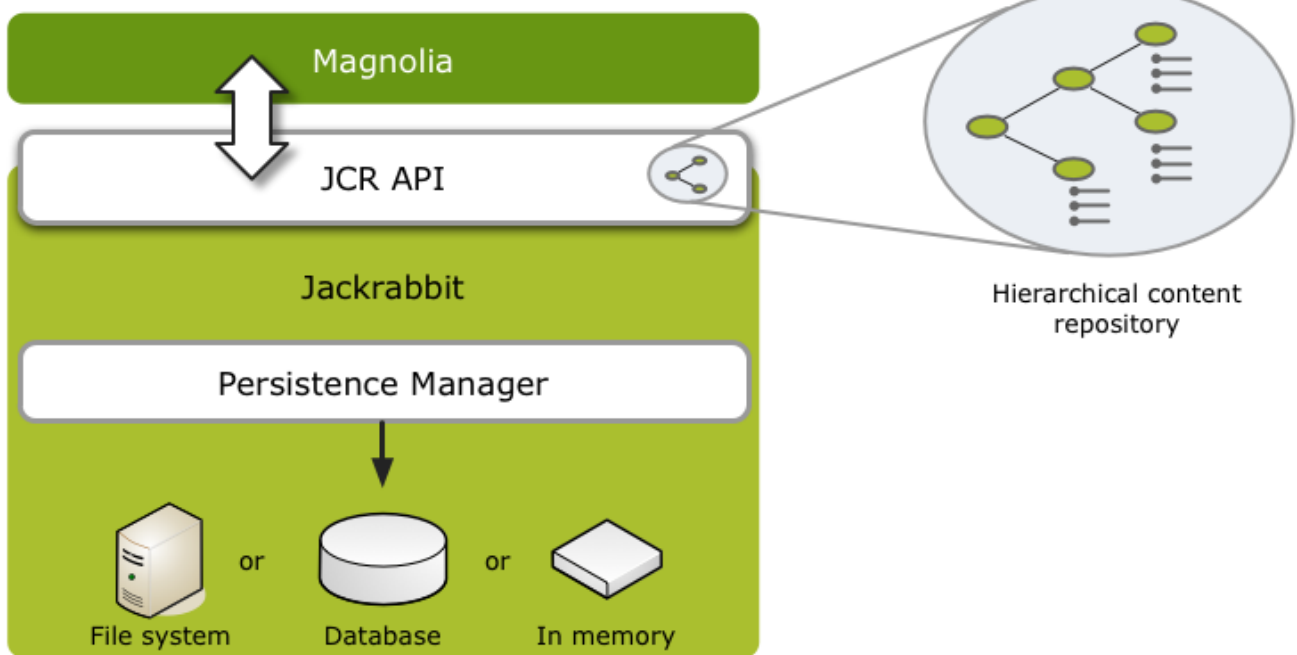
**Summary**

	Tomcat 5.5.27 or higher	JBoss 4.0.5 or higher	Websphere 6.1 SP2 or higher	BEA Weblogic Server 9.2, 10.1 and higher
Linux distributions, kernel 2.6	✓	✓	✓	✓
Windows 2008 Server	✓	✓	✓	✓
Windows 2003 Server SP3	✓	✓	✓	✓
Windows XP SP2, Vista, 7	✓	✓	✓	✓
FreeBSD 7	✓	✓	✓	✓
Mac OS X 10.3 or higher	✓	✓	—	—

## Local database

A relational database is by far the most common persistent storage mechanism in production environments. To ensure fast transactions between Magnolia and the content store the database should be on the same physical machine as Magnolia. For example, if you use MySQL database as persistent store install it on the same server hardware or virtual server as Magnolia.

### Content Storage



Outside of production environments there are other, faster options for persistent store:

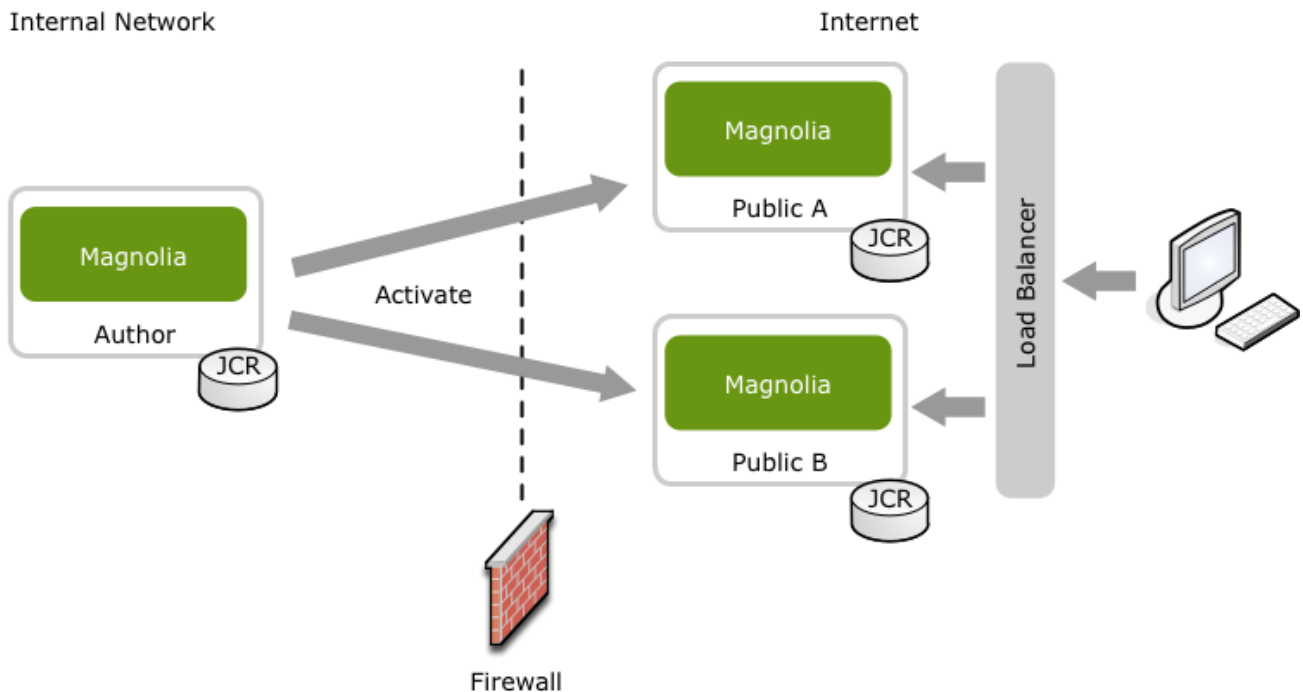
- **File system:** Stores content on the local file system. Not meant to run in production environments, except in read-only cases, but it can be very fast and very easy to maintain.
- **In memory:** This is a great persistence manager for testing and for small workspaces. All content is kept in memory and lost as soon as the repository is closed. Even faster than a file system. Again, not for production use.

## Instances

In Web-facing applications performance is more important at the public-serving end than at the author end. Web content needs to be delivered to visitors without any perceived delay. Magnolia handles this by taking advantage multiple *instances*.

Magnolia is distributed as two web-applications, one acting as the authoring instance and the other as the public environment:

- **Author** instance is where authors work. The author instance activates (pushes) content to public instances.
- **Public** instance receives the activated content and exposes it to visitors on the Web. Public instance resides in a public, reachable location. You can have more than one public instances serving the same or different content which is key to guaranteeing performance and availability.



A distributed instance setup allows you to respond to high availability requirements and sudden increases in traffic. Each of the following problem scenarios can be addressed by adding a new public instance to serve the same content, helping existing instances deal with the load.

- Public instance **failure**. Imagine you have two public Magnolia CMS instances and due to hardware failure one of them is out of operation for a few days. Start a new public instance to replace the broken one.
- Sudden **high load** on your site, also known as [Slashdot effect](#)<sup>1</sup>. Spin off new public instances until the traffic subsides.
- Public instance **blackout**. All public instances are corrupted, broken or compromised and no instances exist to serve content. A small site can deal with this by creating a new public instance and publishing all content to it. Large deployments are more difficult. Some content may have been modified since the blackout took effect and some pages may not be ready to publish yet. In this case the Magnolia CMS [Synchronization module](#)<sup>2</sup> can help. Use the module to activate any previously published versions of content, even if the content was modified further, and skip any pages that were not previously published.

---

1. Slashdot effect [http://en.wikipedia.org/wiki/Slashdot\\_effect](http://en.wikipedia.org/wiki/Slashdot_effect)

2. Synchronization module <http://documentation.magnolia-cms.com/modules/synchronization.html>

## Instance configuration examples

Magnolia CMS configurations range from a minimal one-machine setup to a standard three-instance production setup and beyond.

- **Minimal.** Author and Public instances exist on the same server. This is the default configuration when you install Magnolia CMS on your computer to try it out.
- **Reduced.** Author and Public instances exist on separate servers. The two Public instances are on the same server or one of the Public instances resides on the same server as the Author instance.
- **Standard.** Public instances exist on separate servers. This configuration enables load balancing. Incoming requests can be directed to one server or another depending on availability using hardware or software load balancer.

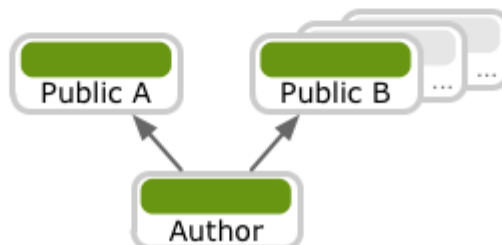
### Minimal



### Reduced



### Standard



## Content authoring and activation

Authoring is the process of writing web content and managing it as part of the website. In Magnolia CMS content is authored in the browser. A common misperception is that content authoring is a performance bottleneck. Adding 50, 100 or 500 authors to a site does not significantly deteriorate system performance. The Java Content Repository can handle a large volume of read and store requests.

In the unlikely event that the author side is underperforming you can use these strategies to spread load:

- **Security.** Segregating editor access to particular sites, site hierarchies or pages prevents too many editors from working on the same content. An editor who can only see the part of the site hierarchy will only edit that part. Permissions are handled through access control lists (ACL) in Magnolia CMS AdminCentral, a standardized well-known practice.
- **Split content to multiple author instances.** This strategy may help in a situation such as site migration where an unusually high number of authors edit content simultaneously. Store content tree `/products` on author instance A and tree `/services` on author instance B, for example. Through virtual URI configuration you can direct editors to the appropriate author instance automatically depending on which tree is their responsibility. Subscriber configuration ensures that content from both author instances is activated to all public instances. **Note!** This strategy can lead to issues such as duplicate content UUIDs or two editors uploading the same binary file since the two author instances don't know about each other.

A more common bottleneck is moving content to public instances. Content is published from the author instance to the public instances via *activation*. Public instances that receive the activated content are known as *subscribers*. Any number of subscribers can subscribe to a single author instance.

Subscribers are key to building high-availability, load-balanced environments since they can be configured to receive targeted content. A subscriber configuration defines the address of the subscriber and the type and location of content it should receive. This way you can map a particular public instance to a very specific part of the author's content hierarchy.

Subscribers can subscribe to any content item: websites, subsections of websites, configuration settings, custom data types and data nodes, forums, comments, documents, images and so on. This means that informed subscriber configuration can alleviate activation bottlenecks also depending on content type (media).

Activating big subtrees of content has a performance impact. Try to manage activations in a piecemeal fashion. In a pinch, turning off versioning can also improve performance as content is version at activation time.

## Clustering

Magnolia CMS can be configured to run in a clustered environment to provide high availability and load balancing:

- **High-availability clusters** are also known as fail-over clusters. Their purpose is to ensure that content is served at all times. They operate by having redundant instances which are used to provide service when a public instance fails. The most common size for a high-availability cluster is two public instances, the standard Magnolia CMS setup. In such a setup the redundant instance may even be dormant (not actively serving content) until it is called to service.
- **Load-balancing clusters** connect many instances together to share the workload. From a technical standpoint there are multiple instances but from the website visitor's perspective they function as a single virtual instance. A load balancer distributes requests from visitors to instances in the cluster. The result is a balanced computational workload among different instances, improving the performance of the site as a whole.

## Jackrabbit clustering

We use Jackrabbit's clustering feature to share content between Magnolia CMS instances. Clustering in Jackrabbit works on the principle that content is shared between all cluster nodes. This means that all Jackrabbit cluster nodes access the same persistent storage (persistence manager and data store).

The persistence manager must be clusterable. Any database is clusterable by its very nature as it stores content by unique hash IDs. However, each cluster node needs its own (private) file system and search index. For more details see [Jackrabbit clustering documentation](#)<sup>3</sup>.

Using Jackrabbit cluster opens up the possibility of cloud-like instance configurations that enjoy cloud benefits such as responsiveness to peak load, reliability and scalability. Note that it is important to create a redundant, second Jackrabbit cluster to avoid single point of failure in the content store.

## Author clustering

Clustering is a high-availability strategy that can be effective on public instances. Multiple public instances can access the same content store simultaneously.

The same setup is problematic on author instances. Authors lock the content store during editing. When one author instance gets a lock on the content store, others cannot edit it.

There are two strategies to work around this limitation. When used in combination they allow you to set up "cluster-like" redundancy on the author side.

1. **Cold standby.** Author A is configured to store data to a local content store. Author B is an identical clone of Author A, except that it is not running. Author B may be shut down or otherwise prevented from accessing and locking content. If Author A suffers an outage, Author B is brought up manually to take its place. This is an effective strategy for instance related issues such as misconfigurations. However, it cannot be used to recover from corrupted content. When content gets corrupted, it immediately corrupts all author instances that try to access it.
2. **Nightly backups.** Take nightly backups of the content store. If content gets corrupted, restore the last known good backup from a disaster recovery system.

---

3. Jackrabbit clustering <http://wiki.apache.org/jackrabbit/Clustering>

While clustering the authoring environment can increase availability for hundreds of concurrently working editors, it has little benefit for disaster recovery since all nodes in the cluster share one single repository and any sort of corruption to the repository will bring down all the cluster nodes.

Even with hundreds of editors working in parallel, each editor only generates load on the instance as they save data or open a page. The editing itself is a purely client-side operation.

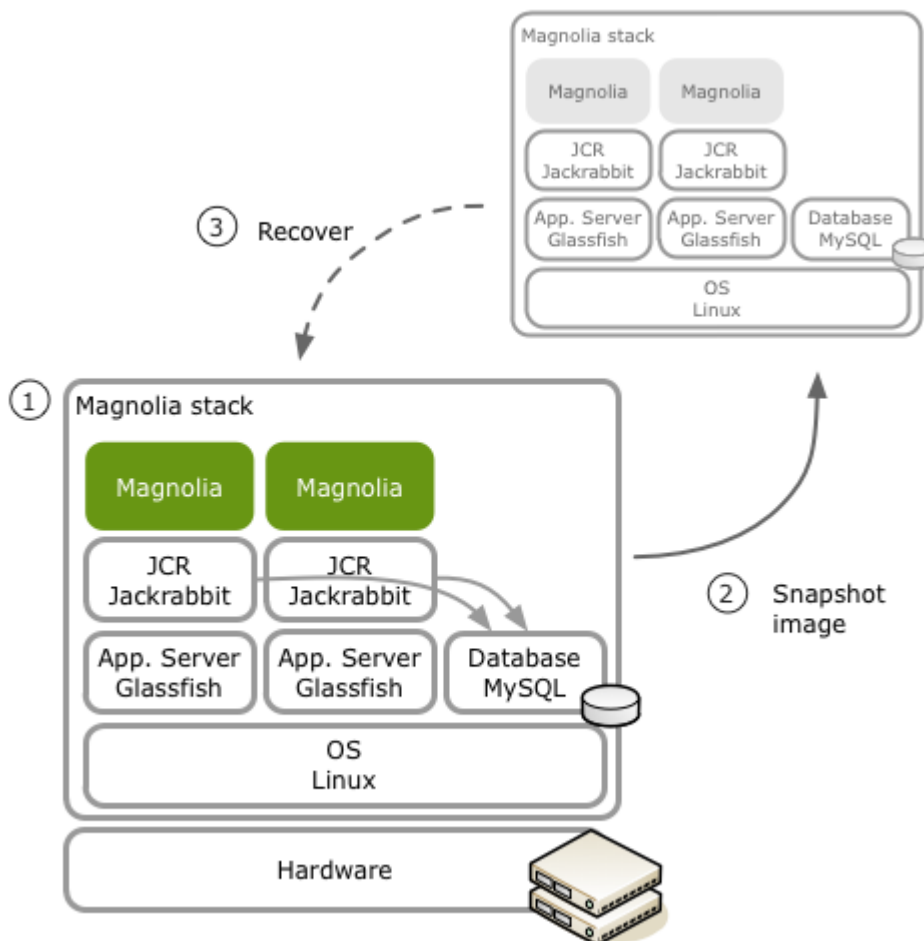
## Disaster recovery and backup

Accidents happen in every system. Servers crash, databases get corrupted and components fail. When that happens a backup can save the day. Implementing a disaster recovery strategy and taking regular backups goes a long way towards ensuring availability.

The recommended disaster recovery strategy for Magnolia CMS is to take periodic snapshots of the entire stack:

1. Run the entire Magnolia CMS software stack in a virtual machine.
2. Take a snapshot image of the whole stack and store it in a disaster recovery system.
3. Recover the image from DR and deploy to a new or repaired server.

The benefits of this approach are speed (15 min for 35 GB snapshot, 5 min for 35 GB recovery) and preserving the current state of the system. Content, templates and resources are in sync when recovered.



Magnolia Network Agreement entitles a licensee to use extra “cold backup” server licenses for the purpose of disaster recovery. A cold server is a server that is turned off until a disaster arises. No other processing or production is done on this server. This way you can create fallback instances that are not in normal use but can be started when a currently-running instances are down.

## Caching

Magnolia CMS employs a web cache to store server responses so that future requests for the same content can be served faster. Using a cache reduces the amount of information that needs to be transmitted across the network, easing the bandwidth and processing requirements and improving responsiveness.

Caching behavior is defined with *policies*:

- **Server cache policy** defines whether the requested content should be cached or not. The standard Magnolia CMS way to make such decisions is with *voters*. Voters are used whenever configuration values are not assigned at start-up but depend on rules. Voters evaluate a rule such as "should content residing at this URL be cached" and return a positive or negative response.
- **Client (browser) cache policy** defines how long the browser may cache a document. The time is passed to the browser in the response header. The default `Never` option instructs the browser to never cache the document. Another option is `FixedDuration` which tells the browser to cache the document for 30 minutes (`expirationMinutes`).
- **Flush policy** defines when to flush the cache. The default configuration observes changes in a repository and flushes the cache if new or modified content is detected.

Magnolia uses [Ehcache](http://ehcache.org/)<sup>4</sup> for its back-end cache functionality. Ehcache is a robust, proven and full-featured cache product which has made it the most widely-used Java cache. You can use another cache engine as long as you implement Java interfaces that allow you to configure caching behavior from AdminCentral.

## Compression

Magnolia compresses content in order to reduce its size during transfer to the client. Compression is a simple and effective way to save bandwidth and speed up your site content delivery. It is a common practice used by Google and Yahoo! for example. ([How to Optimize Your Site with GZIP Compression](http://betterexplained.com/articles/how-to-optimize-your-site-with-gzip-compression/)<sup>5</sup> is a great general introduction to the topic.)

Compression in Magnolia CMS is performed in the `gzip` filter. When a client requests a resource such as `index.html`, Magnolia CMS delivers it zipped. A typical HTML page is compressed to 20% of its original size. So if your page is 100 kB uncompressed, it is 20 kB compressed.

You can configure which content types to compress. By default the `gzip` filter bypasses compression for HTML, JavaScript and CSS because they are explicitly selected for compression in the Cache module configuration. These types can be compressed efficiently because they are text.

---

4. Ehcache <http://ehcache.org/>

5. How to Optimize Your Site with GZIP Compression <http://betterexplained.com/articles/how-to-optimize-your-site-with-gzip-compression/>

## Advanced cache strategies

The Advanced Cache module is an Enterprise Edition only collection of cache strategies to help minimize load on the server while ensuring fresh content is served to users.

- **Serve old content while re-caching.** When using this strategy, cache will not be completely cleaned when content update is detected. All cached entries are retained. When the first request comes in after content update, generation of a fresh cache entry is triggered while further requests for the same entry are served from old cache content until the new entry is ready.
- **Eager re-caching.** When using this strategy, cache will keep a list of most often served entries and attempts to refresh those as soon as an update of content is detected. All other entries are re-cached when requested again, as is the case with the strategy above. The number and lifetime of the most-served content entries is configurable. Statistics about served pages are kept forever by default but you can force Magnolia CMS to clean them after each content update.
- In Enterprise Edition cache can be also configured to **flush only the subtree** to which the activated content belongs. This feature is often used with multi-site support as it is usually not necessary to flush all the sites when activating a piece of content that belongs to any single site.