

# Content Repository Benchmark

## Loading 100 million documents

---



## Goal

The goal of this benchmark is to prove that the Content Repository is scalable enough to be called Enterprise Content Repository.

To achieve this goal, we load at least 100 million office documents into a single Content Repository, using commodity hardware and measure the performance of the system during and after load is complete.

We wanted to beat Alfresco, our only open source competitor that published any similar benchmarks.

For this, we have built a Benchmark Tool, that can be downloaded by anyone to repeat the benchmark.

## Goals in numbers

Our goals are in numbers that can be measured:

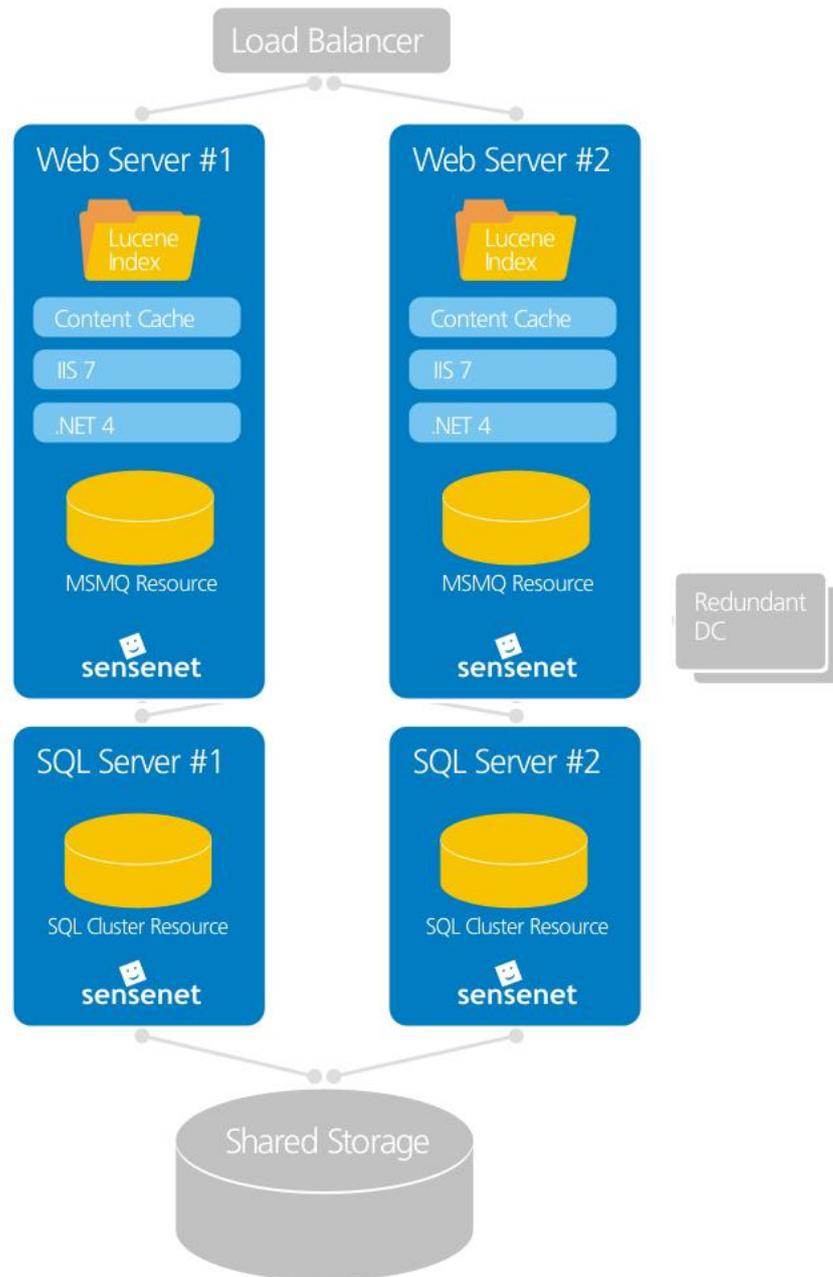
- Number of documents: 100 million
- Type and size of documents: 50 documents similar in type and size to the ones Alfresco used
- Load rate: at least 150 content per second
- Single document retrieval time: under 1 second

## Summary of results

In the final round we loaded 100 million documents into 20 million folders, with an average load throughput of 300 content items per second, while load time for a single content remained under 1 second on the UI.

## Setup

The content repository is set up in High Availability mode, except for the SQL server that was not in Failover mode. We used one workstation to drive load, running the Benchmark Tool. Another workstation was used to monitor the benchmark. The load was driven to the ASHX handler of the benchmark, simulating a REST API, and files were uploaded to the Content Repository via HTTP.



## Software

We used

- Version 6.1 of the Content Repository
- SQL Server 2012
- Windows Server 2008 R2 SP1 x64

## Hardware

IBM Innovation center provided the following hardware environment for us:

Blade chassis:

- IBM BladeCenter H

Blade servers:

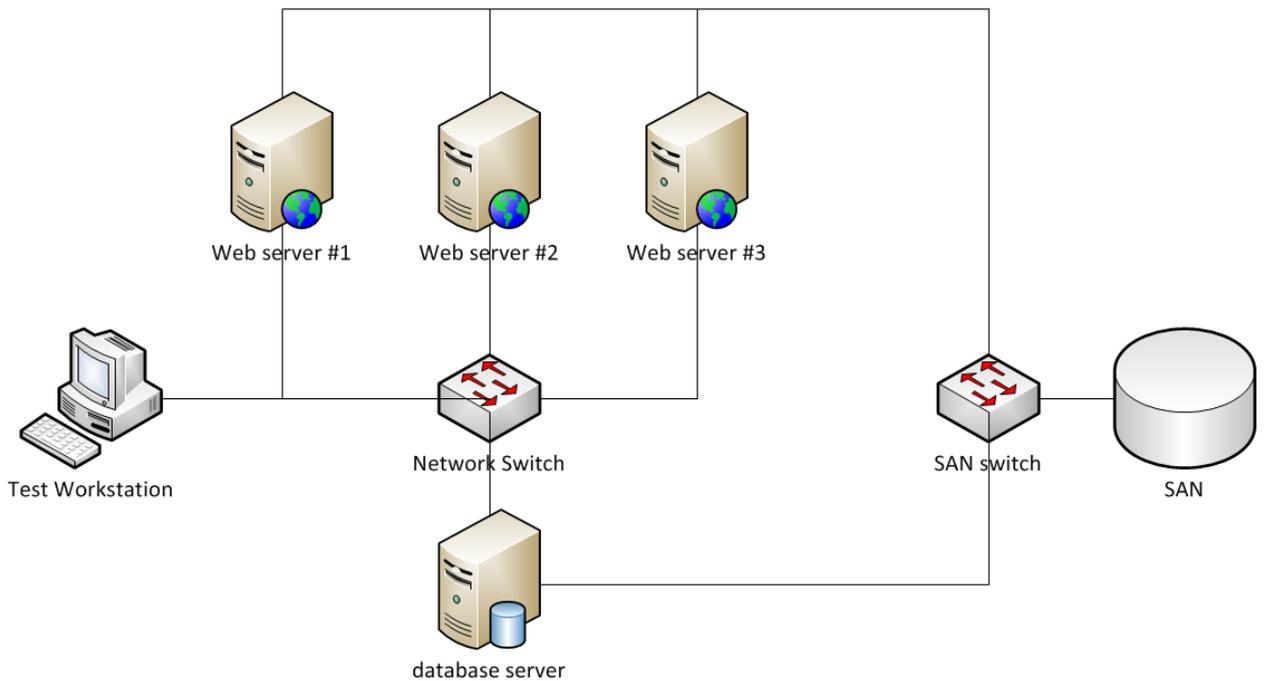
- Web servers (3pcs):
  - IBM HX5 Blade
    - CPU: 2 x Intel Xeon Processor EX L7555 1.86GHz, 24MB Cache, 8 Core + Hyper Threading
    - RAM: 64Gbyte RAM
    - Disk (internal): 2 x 50Gbyte SSD Raid-0 (stripe)
    - Storage (SAN): 500Gbyte-os SAN volume
    - Network: 1Gbit Ethernet
    - Fiber Channel: 2x8Gbit FC
    - Operating System: Windows Server 2008R2 SP1 x64
    - Web szerver: IIS7.5
    - .Net keretrendszer: .Net Framework 4.0
    - Terheléselosztás: Microsoft Network Load Balancing
- SQL server (1pcs):
  - IBM HX5 Blade
    - CPU: 2 x Intel Xeon Processor EX L7555 1.86GHz, 24MB Cache, 8 Core + Hyper Threading
    - RAM: 96Gbyte RAM
    - Disk (internal): 2 x 50Gbyte SSD Raid-1 (mirror) tömbbe szervezve
    - Storage (SAN): 4,5Tbyte-os SAN volume
    - Network: 1Gbit Ethernet
    - Fiber Channel: 2x8Gbit FC
    - Operating System: Windows Server 2008R2 SP1 x64
    - SQL server: Microsoft SQL Server 2012

Storage:

- IBM Storwize V7000
  - HDD: 20 x 600Gbyte SAS 15.000RPM HDD Raid-10
  - SSD: 2 x 200Gbyte Enterprise SSD
  - Fibre Channel: 8x8Gbit FC

Easy Tiering was switched on; access of frequently accessed data is from RAM and SSD.

## Logical network diagram



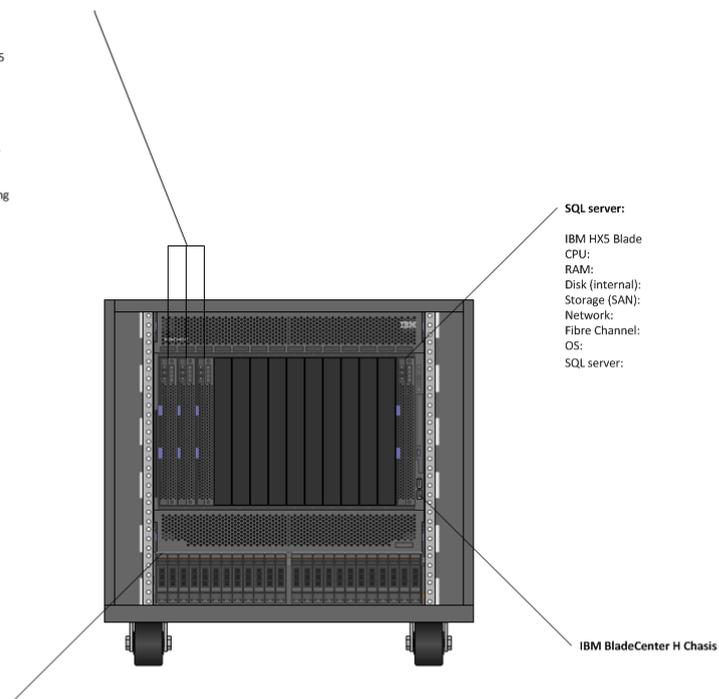
## Blade configuration

### Web servers:

IBM HX5 Blade  
 CPU: 2 x Intel Xeon Processor EX L7555  
 RAM: 64Gbyte RAM  
 Disk (internal): 2 x 50Gbyte SSD Raid-1 (mirror)  
 Storage (SAN): 500Gbyte SAN volumes  
 Network: 1Gbit Ethernet  
 Fibre Channel: 2x8Gbit FC  
 OS: Windows Server 2008R2 SP1 x64  
 Web server: IIS 7.5  
 .Net Framework: .Net Framework 4.0  
 NLB: Microsoft Network Load Balancing

### SQL server:

IBM HX5 Blade  
 CPU: 2 x Intel Xeon Processor EX L7555  
 RAM: 96Gbyte RAM  
 Disk (internal): 2 x 50Gbyte SSD Raid-1 (mirror)  
 Storage (SAN): 8 Tbyte SAN volume  
 Network: 1Gbit Ethernet  
 Fibre Channel: 2x8Gbit FC  
 OS: Windows Server 2008R2 SP1 x64  
 SQL server: Microsoft SQL Server 2012



### Storage:

IBM Storwize V7000  
 HDD: 20 x 600Gbyte SAS 15,000RPM HDD Raid-10  
 SSD: 2 x 200Gbyte Enterprise SSD with Easy tiering on  
 Fibre Channel: 8x8Gbit FC

# Benchmark Tool

## How it works

The Benchmark Tool consists of

- a console application that sends requests to the Sense/Net ECM servers,
- and an HTTP handler that has to be installed on every server node of the installation.

The console application sends the following requests to the handler:

- a request is sent to the handler to create a Folder in the Content Repository,
- a file is sent to the handler to store it in the Content Repository.

The Benchmark Tool will send multiple requests simultaneously in a way that a pre-defined folder structure will be created in the Content Repository with the leaf folders containing a pre-defined set of benchmark files. The benchmark files that will be stored multiple times in the Content Repository are initially located in the file system under a configurable path that can be accessed by the console application. While running, the console application sending requests and files to the servers logs the request infos and response times and displays info about the actual and average throughput of the Sense/Net installation including but not restricted to the average CPS (Content per Second) data. The Benchmark Tool is able to send requests to multiple web nodes simultaneously so high availability setups can be benchmarked without the need of a network load balancer to be configured.

## Setup

To set up your Sense/Net installation, please read [High availability](#) and [Hardware Requirements for Production Environment](#). Once you have your servers up and running, commence with the following steps:

- copy the **benchmark.ashx** into the webfolder (the benchmark.ashx is included in the Benchmark project in the [source package](#)),
- prepare a client computer for running the Benchmark Tool
  - this computer does not need to have server-version Windows installed, Windows 7 is also adequate,
  - the performance of this client computer may affect performance measurements, so use a computer with adequate resources,
  - recommended configuration is detailed in the [#Test results](#) section,
- copy the **SenseNet.BenchmarkTool.exe** and **SenseNet.BenchmarkTool.exe.config** files to an arbitrary location on the client computer,
- prepare a set of files on the client computer to be used for benchmarking:
  - we recommend using the reference file set to be able to make comparisons to our reference measurements (see [#Test results](#)).

Before doing any measurements you also need to configure benchmark parameters as detailed in the next section.

## Configuration of benchmark tool

To configure benchmark parameters edit the config elements in **SenseNet.BenchmarkTool.exe.config**:

- **ThreadCount**: defines the maximum number of concurrent requests.
- **MaxFileCount**: benchmarking will stop if the maximum file count is reached.
- **FolderProfile**: semi-colon separated list of number of folders on different levels, starting with the number first level folder structure items. See [#Folder profile](#) for details.
- **FileRootPath**: the file system path of the folder containing the files to be used for benchmarking.
- **ContentRepositoryPath**: the Content Repository path for the root of the folder structure to be created. This path is automatically deleted upon execution if exists and is also automatically created.
- **Urls**: semi-colon separated list of server node url addresses. You may include an url more than once resulting in a heavier load on the server of the repeated url. Urls are requested using round-robin schema, but load on individual servers is also affected with the *EnableEqualLoad* setting.
- **SessionName**: an arbitrary name of the benchmark session to be included in log files for easier catalogizing of benchmark logs.
- **SaveDetailedLog**: set it to false to switch creation of detailed log off. Detailed log can grow very large for long-running benchmarks.
- **EnableEqualLoad**: if true the number of requests sent to a specific address will not reach the quota defined by the proportions of the different urls. For example if *ThreadCount* is 300 and urls are server1;server1;server2, then server1 will receive 200 simultaneous requests and server 2 will receive 100 simultaneous requests. If this is set to false it may happen that the load on one server node tops to the total number of threads resulting in drop of load on other nodes and thus an inoptimal benchmark situation.

## Folder profile

The *FolderProfile* setting defines the folder structure to be created in the Content Repository. This affects benchmark measurement results as the number of requests creating folders or files waiting for a dependent request to finish may vary from scenario to scenario. Below is an example of the structure created with 2;3 set as *FolderProfile* and */Root/YourDocuments/Benchmark* set as *ContentRepositoryPath*- here 2 folders will be created on top level and 3 folders for every top level folder. Files will be created in the leaf folders:

- `/Root/YourDocuments/Benchmark`
  - Folder-0
    - Folder-0
      - File-0
      - ...
    - Folder-1
    - Folder-2
  - Folder-1
    - Folder-0
    - Folder-1

- [Folder-2](#)

## Configuration of Content Repository being benchmarked

A system to benchmark can be set up the usual ways. In our tests we used a [high availability](#) setup with 3 web nodes. To see how you can setup Sense/Net ECM with NLB read the following article:

- [How to configure Sense/Net in NLB](#)

In order to achieve optimal results we recommend to change some system settings before running the benchmark. One important thing is to configure the Lucene indexing merge policy for frequent indexing writes. Here is an example you can use in the web.config appsettings:

```
<appSettings>
  <add key="CacheContentAfterSaveMode" value="Containers" />
  <!-- MergeFactor. Default is 10. For write-intensive scenarios set it to 100. -->
  <add key="LuceneMergeFactor" value="100" />
  <!-- RAMBufferSizeMB. Default is 16.0. For write-intensive scenarios set it to 48.0. -->
  <add key="LuceneRAMBufferSizeMB" value="48.0" />
  <!-- MaxMergeDocs. Default is Int32.MaxValue. For write-intensive scenarios set it to
100000. -->
  <add key="LuceneMaxMergeDocs" value="100000" />
```

Another important setting is the configuration of content cache for optimal memory usage. Both content caching after saving and the memory limit of cache needs to be set. For content caching after saving we used the caching of the containers since all folders imported will be loaded more than once whereas files will only be saved and never loaded during the benchmark:

```
<add key="CacheContentAfterSaveMode" value="Containers" />
```

Setting the memory limit of the cache can also be necessary should the ECMS run out of memory. You can read about this in the following article:

- [Content cache](#)

## Performance monitoring

While the Benchmark Tool is running it displays the following information about the current and average performance/throughput of the servers under load:

```
22      0/1111  34/572  1,00/21,00MB  Q:50388
      CPS: 34/76,00
      F/F: 0,00      MBps: 0,00      T:40
```

Output of the Benchmark tool - info is displayed in every second

- (1) elapsed seconds, (2) folders created (last second) / total folders created, (3) files created (last second) / total files created, (4) MBytes uploaded (last second) / total MBytes uploaded, (5) total number of files queued for upload
- (6) current CPS (last second) / total average CPS
- (7) number of files / number of folders (last second), (8) total average MBps, (9) number of requests sent

## Log files

The following log files are created during execution:

- **BenchmarkLog-<session name>-<start time>-summary.csv**: a per-second based summary of benchmark with similar data as displayed to console by the Benchmark Tool, including actual file/folder count, total file/folder count, total average CPS, etc.
- **BenchmarkLog-<session name>-<start time>-details.csv**: a detailed log containing all requests with request-specific info like request url path, content type (Folder/File), response time, total duration, request success, etc.
- **BenchmarkLog-<session name>-<start time>-error.csv**: a text file containing all error messages received from the web servers also including the uri and timestamp of the request that resulted in the error.

## Performance counters

The Sense/Net Benchmark Tool introduces the following performance counters that can be used in *Performance Monitor* (*perfmon.exe*) on the client machine running the Benchmark Tool:

- **CPS**: current CPS (content per second)
- **CPSAVG**: total average CPS
- **CPSAVG20**: average CPS for the last 20 seconds

## Process

We first created the benchmark tool, and tested it with 1 million documents on local smaller hardware. When the tool was deemed complete, we ran the benchmark in the IBM Innovation center in several iterations.

We then run iterations of loading 2, 15, 45, 55, 80 and 120 million pieces of content into the Content Repository. The 120 million one loaded 100 million documents into 20 million folders, with an average load throughput of 300 content per second.

The following configuration was used in the final benchmark.

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="ThreadCount" value="150"/>
    <add key="MaxFileCount" value="120000000"/>
    <add key="FolderProfile" value="100;100;10;24"/>
    <add key="FileRootPath" value="c:\benchmarkfiles"/>
    <add key="ContentRepositoryPath" value="/Root/YourDocuments/Benchmark"/>
    <add key="MaxTaskQueueSize" value="100"/>
    <add key="Urls"
value="http://server1/benchmark.ashx;http://server2/benchmark.ashx;http://server3/benchmark.ashx" />
    <add key="SessionName" value="3machines-120M-150threads"/>
    <add key="SaveDetailedLog" value="false"/>
  </appSettings>
</configuration>
```

```
<add key="EnableEqualLoad" value="true"/>
</appSettings>
<startup><supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.0"/></startup>
<system.net>
  <connectionManagement>
    <add address="*" maxconnection="2000000" />
  </connectionManagement>
</system.net>
</configuration>
```

## Results

We have conducted a benchmark on IBM servers on 2012.05.29. We used a 3-webnode NLB setup, a separate SQL server and a SAN for storage. Our goal was to import 100 million files into one single Sense/Net Content Repository, with all content indexed and immediately searchable after and during the benchmark.

- **Elapsed time:** 336909 sec (3 days, 21 hours, 35 mins, 9 secs)
- **Number of folders imported:** 24 24 259
- **Number of files imported:** 100 000 005
- **Total size of files:** ~4 TB
- **Database size:** ~8 TB
- **Average file size:** 40 039 bytes
- **Average content/sec:** 304
- **Average MBytes/sec:** 11

Performance of the UI was slower than with 1 million content, but loading a single content remained under 1 second.

## Appendix 1 – List of files

<b>File name and type</b>	<b>File size (bytes)</b>
sensenet.log	3 745
PriceList.txt	8 538
filesize-averages.xlsx	15 461
2627.jpg	21 546
1697.jpg	22 187
1436.jpg	22 954
1728.jpg	24 855
0702.jpg	25 197
1655.jpg	25 202
1336.jpg	25 403
2286.jpg	25 947
1651.jpg	26 012
1349.jpg	26 521
0545.jpg	26 645
9337.jpg	26 661
4116.jpg	27 080
4483.jpg	27 631
0731.jpg	27 693
4937.jpg	27 808
1299.jpg	27 990
2574.jpg	28 391
2608.jpg	28 687
0210.jpg	28 976
1209.jpg	29 612
testdoc (4).doc	29 696
TrainingCurriculum.docx	30 287
OrderForm.xlsx	34 005
LabContentPresenter.docx	39 726
LabQueryIndexing.docx	41 889
LabWorkflow.docx	42 176
testdoc (3).doc	44 544
testpdf (4).pdf	45 071
BDM-1-pager.docx	45 832
testpdf (8).pdf	48 688
testpdf (9).pdf	49 168
testdoc (6).doc	50 176
testpdf (3).pdf	51 171
testpdf (6).pdf	51 171
testpdf (7).pdf	51 171
testpdf (1).pdf	51 433
OrderForm.xls	52 224
SenseNetEnterpriseLicensing.doc	53 760

testpdf (2).pdf	56 724
SenseNet_System_Requirements.doc	61 952
testdoc (5).doc	62 464
BDM-2-pager.docx	69 175
testdoc (2).doc	70 144
testdoc (1).doc	70 144
testdoc (7).doc	70 144
SenseNetBrochure.docx	78 772
testpdf (5).pdf	84 961