

Joget Workflow Deployment Best Practices

- [Installation as a Service](#)
- [Java VM Configuration](#)
 - [Java VM Tuning](#)
- [Web Application Server Configuration](#)
 - [Apache Tomcat Configuration](#)
 - [Linux Configuration](#)
- [Database Configuration](#)
 - [Database Indexing](#)
 - [MySQL Configuration](#)
- [Clustering and Load Balancing](#)
- [Logging and Troubleshooting](#)
- [Backup and Restore](#)
 - [Backup Data](#)
 - [Restoring Data](#)

The default Joget installation comes with minimal performance configuration as it aims to cater for installation on small machines, e.g., a local development PC.

For production deployment on a server, the Java Virtual Machine (JVM), application server, database and operating systems should be tuned for performance. Actual real world performance depends on many factors. These include the deployment architecture, server and network specifications, complexity of the processes/apps, actual usage patterns by different users, etc.

This document presents some of the best practices in terms of performance tuning and optimization to get the best out of your deployment.

Important Note: These recommendations help to serve as general tips and guidelines, but in actual practice it would depend on each deployment's unique environment. There are also many resources available online to help tune performance for Java, application servers and databases.

Installation as a Service

For production deployments, you may want to install the web application server, e.g., Apache Tomcat, as a service. Please refer to the documentation on the relevant operating system to do so. A guide to install Tomcat as a Windows service is available at [Installing Joget as a Windows Service](#).

Java VM Configuration

Java VM Tuning

An important configuration would be the JVM memory allocation. If this is too low, the system will run out of memory. However if the setting is too high, there might be quite a large overhead in garbage collection. To get an optimum setting might require a bit of trial and error sometimes, depending on the usage environment.

Here are the steps to get started. You may want to start with a 1GB max heap setting and increase if the need arises.

1. Stop Joget.
2. Open and edit [KB:Joget Installation Directory]/joget-start.bat.
3. Modify

```
1.
set JAVA_OPTS=-XX:MaxPermSize=128m -Xmx512M -Dwflow.home=./wflow/
```

on line 13 to

```
1.
set JAVA_OPTS=-XX:MaxPermSize=256m -Xmx1024M -Dwflow.home=./wflow/
-XX:+UseConcMarkSweepGC
```

4. Save and start Joget.

NOTE:

Please take note should you encounter any of the following OutOfMemory errors:

java.lang.OutOfMemoryError: Java heap space	Increase the maximum heap size -Xmx
java.lang.OutOfMemoryError: PermGen space	Increase the maximum permgen size -XX:MaxPermSize
java.lang.OutOfMemoryError: GC overhead limit exceeded	Enable the concurrent garbage collector -XX:+UseConcMarkSweepGC

There are many online resources for further tuning, e.g., <http://www.oracle.com/technetwork/java/performance-138178.html>.

Web Application Server Configuration

Apache Tomcat Configuration

For Apache Tomcat, try setting the maxThreads Connector thread pool settings based on the estimated request load. For example, to set the maximum number of threads to 1000, edit tomcat_directory/conf/server.xml. Locate the line

```
<Connector URIEncoding="UTF-8" port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

and add a maxThreads="" such as

```
<Connector URIEncoding="UTF-8" port="8080" protocol="HTTP/1.1" maxThreads="1000"
    connectionTimeout="20000"
    redirectPort="8443" />
```

This would require a Tomcat restart to take effect.

There are many resources available online, e.g., <http://www.itworld.com/networking/83035/tomcat-performance-tuning-tips>.

Linux Configuration

Linux / Unix systems have a limit on the number of files that can be concurrently opened by a process. When the limit is reached, there will be an exception "Too many open files". The default for most distributions is 1024, which will not be enough for high traffic deployments. In Linux, increasing the ulimit

```
ulimit -n 4096
```

works as a workaround but this is only valid for the session. For permanent configuration, please refer to documentation for the specific distribution, e.g., <http://posidev.com/blog/2009/06/04/set-ulimit-parameters-on-ubuntu/>.

Database Configuration

Database Indexing

Form data tables generated and managed by Joget are prefixed with **app_fd**. By default, primary keys and foreign keys are automatically generated for these tables where necessary.

However, in more complex apps, there would be more complex database queries that make use of these tables. As such, it is recommended for indexes to be added manually into table columns when required.

MySQL Configuration

When using InnoDB in MySQL, setting an appropriate InnoDB Buffer Pool is important for large-scale performance. Details can be found here: <http://dev.mysql.com/doc/refman/5.0/en/innodb-buffer-pool.html>.

In environments with large data and queries, one way to improve performance is to use the **query cache**. This can help speed up query performance tremendously. Details can be found at <http://dev.mysql.com/doc/refman/5.0/en/query-cache.html>.

You may need to increase the number of **maximum connections** allowed if there are a lot of concurrent requests. <https://dev.mysql.com/doc/refman/5.0/en/too-many-connections.html>

It is possible to identify slow queries in MySQL by enabling the slow query log. This helps to identify areas for possible indexing and optimization. <http://dev.mysql.com/doc/refman/5.0/en/slow-query-log.html>

Clustering and Load Balancing

For large scale and high-availability deployments, clustering and load balancing may need to be implemented using the **Large Enterprise Edition**. A high-level description is as follows:

The main configuration is actually done at the web application server, in this case Tomcat. There's a quite comprehensive documentation for this at <http://tomcat.apache.org/tomcat-6.0-doc/cluster-howto.html>.

You'll need a load balancer as well. This could be hardware or software, and it's possible to use Apache web server for this purpose. Using the `mod_proxy_balancer` module (http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html), traffic is directed to various Tomcat nodes. Depending on configuration, the approach could be load balanced or failover, and sticky sessions are recommended.

The database and wflow files (configuration/uploaded files) would also have to be shared from a centralized server.

Logging and Troubleshooting

When running Apache Tomcat, logs are stored in the **tomcat_directory/logs** directory. In particular, the files **joget.log**, **catalina.out**, and **localhost.yyyy-MM-dd.log** capture information and errors that are generated.

Should you encounter any issues or errors, it is best that the following information is provided when reporting the issue to Enterprise Support or the community forums:

- Steps to reproduce the issue
- Error messages (including any possible JavaScript errors) in the browser
- Copy of the log files mentioned above
- Screenshot(s) showing the problem
- Sample app that produces the issue

The more information provided, the faster an issue can be identified and resolved.

Backup and Restore

Backup Data

To backup an installation:

- Backup all configuration files and uploaded data files stored in **joget_directory/wflow**.
- Backup the database. e.g., in MySQL you can use the **mysqldump** utility.

Restoring Data

To restore data to an installation:

- Restore all configuration files and uploaded data files stored in **joget_directory/wflow**.
- Restore the database.