MoSKito FAQ

What's the general project status?

General status

Production quality and being continuously developed further ;-)

History
MoSKito was started in 2007 and used in various production environments (friendscout24, c-date.com, parship.de, etc) since 2008. So, today it is very stable and absolutely ready for production.

In the beginning of 2013, we extracted some MoSKito parts into separate projects, namely **MoSKito-Control** and **MoSKito-Central**. The "original" MoSKito, the part with the core functionality, has been renamed to **"MoSKito-Essential"** (just because it needed a name, and MoSKito-Core was already taken) ;-)

Know more about the way we treat MoSKito Projects and Components.

MoSKito-Essential

Status: Stable & production ready.

The project is in production since 2008, being continuously enhanced since then (support for *aop* and *spring*, *cdi*, etc).

MoSKito-Control

Status: In production, but still beta (mostly due to fine-tuning).

MoSKito-Control was initially created as part of a customer's project and to be later donated back to MoSKito project. The story is covered in *Everything in Control* blog port.

After the donation, a huge part of the code has been rewritten for making it more generic and removing the customer-specific code. Currently, MoSKito-Control is in **beta**.

MoSKito-Central

Status: Partially released, some parts are still in beta (support for *postgres* and other features).

MoSKito-Central has been a part of MoSKito, and then refactored into a standalone project in 2013. It got some recent additions, for example, *psql* or *mong* (currently in progress).

MoSKito on mobile devices

There are apps for MoSKito-Essential and MoSKito-Control in the iOS app store. They are free and fully functional.

MoSKito-Control for android is being currently developed by a volunteer.

How big is MoSKito's performance overhead?

The performance overhead, caused by MoSKito on your system, depends on many factors or, better say, values.

First of all, it's the integration type and number of monitoring points. To be clear: each monitoring point costs time, the exact time depends on integration type.
For example, Proxy integration is slower than AOP compile time weaving.

However, in almost all cases, the costs are in lower microseconds area, depending on your hardware.

For most web applications, the performance drawback of MoSKito is not measurable. It may consume a bit more CPU, but the end user will most likely not notice it.

In other words, you can use MoSKito without thinking about performance impacts, until you really see some.

What's MoSKito's memory footprint?

MoSKito-Essential runs within your app's heap space and therefore consumes some of it.

Usually, the amount of memory used by MoSKito is significantly lower than the amount of memory used by the monitored code. So, in case you have millions of monitored classes and your heap size is above 10 Gb, MoSKito will consumer some few hundred MBs, which is about 1-3 percent of your heap.

If you have a small application, with 128 MB heap, MoSKito will usually consume 1-3 MB. You can calculate your exact memory footprint by creating a histogram.

Are there any external reviews and usage references for MoSKito?

We have some feedbacks in our blog, as well as reviews or feedbacks from conferences and stuff. There are also some references on the homepage http://www.moskito.org.

We can also help to contact MoSKito users from other companies.

Is MoSKito-Control a standalone solution or it requires Essential & Central?

From the deployment point of view, MoSKito-Control is a standalone solution. But it doesn't collect any performance data by itself, instead it collects the performance data gathered by other applications. One of such applications is **moskito-control-agent** for Java, which is deployed into the target app (via *maven* dependency or just by adding the lib). MoSKito-Control uses **MoSKito-Essential** to actually gather the statistics.

So, the data flows like this:
MoSKito-Control ---> network ---> agent ---> local_call ---> MoSKito-Essential.

Technically, a MoSKito-Control agent can be provided for any other language or environment, such as PHP or .NET, however, there are no supported 3rd party languages yet.

This blog post describes all MoSKito projects and components, explaining their relations. We understand it can be confusing in the beginning, but we were not able to come out with shorter naming yet ;-)
Well, after all we are mainly developers and not marketers.

What framework is used for charts? Do charts support zooming in/out and ranges?

We use Google Charts: https://developers.google.com/chart/. Please refer to Google charts documentation for more details.

In the iOS apps we have a small 3rd party open source engine, because Google charts were too heavy to use (especially to load on lower bandwidth).

Can MoSKito-Central store information in Oracle ? What instruments are used for that?

We using **Hibernate 4.2.0** via **JPA 2.0**.
This should also support Oracle by just configuring the proper *jdbc* driver.

However, MoSKito-Central is designed to support custom storages, so it is not a problem to add a customised storage that writes data in the database and format of your choice, by implementing an interface. You can also get this implementation from us ;-)

Does MoSKito support HTTPS for communicating with WebUI and other components?

Yes, it does.

All web-based MoSKito applications support HTTPS (or run in a container on HTTPS port).
Also, all the iOS apps support basic HTTP authentication.

Can I get support for MoSKito?

Yes, the options are listed on the Support page.

For professional support, please write to moskito@anotheria.net.

Where is the source code?

Until recently, we had the source code in our own subversion repository, accessible from outside.

However, we migrated it all to Github:

https://github.com/anotheria/moskito

https://github.com/anotheria/moskito-control

https://github.com/anotheria/moskito-control-agent

https://github.com/anotheria/moskito-central

https://github.com/anotheria/moskito-examples

https://github.com/anotheria/moskito-demo

Downgraded privileged process is unable to read /proc/self/fd

Yes, there is a bug and a solution: **MSK-160** - Getting issue details... STATUS

How do I get rid of the logging in my rootLogger with Log4j/Log4j2

Actually you just have to define proper Logger in your log4j configuration.

If you are only using the moskito-core components you will need a Logger with the name ***moskito.***

For example in Log4j2 it looks like the following:

Log4j2 example

```
<!-- Create Appender -->
<RollingRandomAccessFile name="MOSKITO" fileName="${logDir}Moskito.log" filePattern="${backupDir}Moskito.log-%d
{yyyyMMdd}${compressionType}">
                   <!-- Put you own pattern and policy here -->
                        <PatternLayout pattern="${defaultLayout}" charset="${patternCharset}"/>
                        <Policies>
                <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
            </Policies>
</RollingRandomAccessFile>

<Logger name="moskito" level="INFO" additivity="false">
            <AppenderRef ref="MOSKITO"/>
</Logger>
```

If you are also using the HTTPEndpoint of Moskito, you furthermore have to define an Logger for ***"org.moskito"*** and ***"net.anotheria"***.

```
<Logger name="org.moskito" level="ERROR" additivity="false">
            <AppenderRef ref="MOSKITO"/>
</Logger>

<Logger name="net.anotheria" level="ERROR" additivity="false">
            <AppenderRef ref="MOSKITO"/>
</Logger>
```

Can I monitor multiple webapps in same tomcat instance and how.

For embedded integration it was always possible, since MoSKito is integrated into the target webapp.

For remote integration it is was not possible because each webapp would need an rmi port, and it was not possible to specify a different port per webapp.

It became possible with version 2.7.4, as fix of: DISTRIBUTEME-33 - Getting issue details... STATUS ,
MSK-324 - Getting issue details... STATUS .

So to enable it you will need:

1. Update to 2.7.4 (or above)
2. Add following to your web.xml (different port for every web.xml of course, port 9450 is just an example).

```
<context-param>
    <param-name>moskitoRmiPort</param-name>
    <param-value>9450</param-value>
</context-param>
```

You will also require DistributeMe 2.2.5 or above, which is a dependency of MoSKito 2.7.4, but in case you specified a direct dependency of DistributeMe for other reasons in your code, please upgrade it to 2.2.5 at least.

How to switch off embedded Olark chat in MoSKito Inspect

If you want to switch off Olark chat in MoSKito inspect you can simply set "*showOnlineHelp*": *false* in *moskito-inspect.json*.

```
"showOnlineHelp": false,
```