MoSKito-Essential Overview

After reading this section, you will know...
... MoSKito definition and the features that separate MoSKito from other Java monitoring solutions.

On this page

Definition

MoSKito is a multi-purpose, non-invasive, interval-based monitoring system for collecting, storing and providing instant analysis of application's performance and behavior data.

MoSKito **simplifies** your efforts to measure and optimise the performance of your application and makes your architectural decisions **visible** and **measurable**.

Features

Collect any performance data

MoSKito is multi-purpose, it can collect the widest range of statistical data. As mentioned in Concepts, MoSKito gets data from producers, and these can literally be everything.

What is *everything*? Threads. Memory. Caches. Storages. Services. Averages. ExecutionTime. RequestCount. CPU Usage. IOStats. Registrations. Payments. Conversion. Partner Performance. Online Users. ScamScores. Load distribution. User-dependent load (guest/member). Call traversal. Path-dependent measurement... Whatever you need!

Both Business and Performance indicators can be monitored. MoSKito gathers all the essential info in one place.

No code change required

MoSKito is non-invasive. This means it doesn't require painful integration or any changes in your app's code.

MoSKito is pluggable into any application via *filters*, *proxies*, *annotations*, *inheritance*, *AOP*, *class loaders* or *plain calling*.

MoSKito measures *around* the monitored code, without changing it. Thus, there's no additional impact on your app's performance while monitoring.

Read more about getting MoSKito, configuring and integrating it into your application.

See more with short intervals

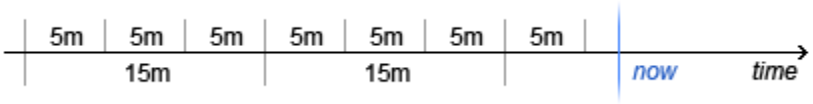
MoSKito is interval-based, it takes a snapshot (counts activities of a monitored producer) every:

- 1 minute,
- 5 minutes,
- 15 minutes,
- 1 hour,
- 12 hours,
- 1 day (24 hours).

This means, at any moment you may know the producer performance for the last 1 minute, 5 minutes, etc. Why is that so?

The behaviour of a live system depends on million things: hour, weekday, weather, holidays and good karma. To have a full picture, you have to consider even the smallest factors.

When you collect large amounts of data, you may easily miss the anomaly that is influencing your system. But when you gather data within short intervals, you may easily compare it with similar periods of time (mornings, Mondays and so on). Working with shorter periods always give more understanding about system's behaviour.



Keep your data private

Many good monitoring services, effective and easy-to-set-up, use cloud solutions. This means all the performance info from your app is being processed on the 3rd party servers.

It's ok until privacy issues come into focus. What if sharing performance data contradicts to your security policy?

MoSKito does not use any external servers or resources for processing or storing data. It is deployed on your local machine, and the collected data stays here as well: in app's memory or within a personally configured storage place. This solves all security and privacy issues.

Define and watch critical areas

Within your application, some parameters might be more important for you than the others. You might also want to monitor them 24/7, knowing at a glance if they are doing well.

With **thresholds**, you may continuously monitor a single parameter (producer) and receive a signal when the performance of this parameter goes beyond the boundaries that you've set for it.

System state

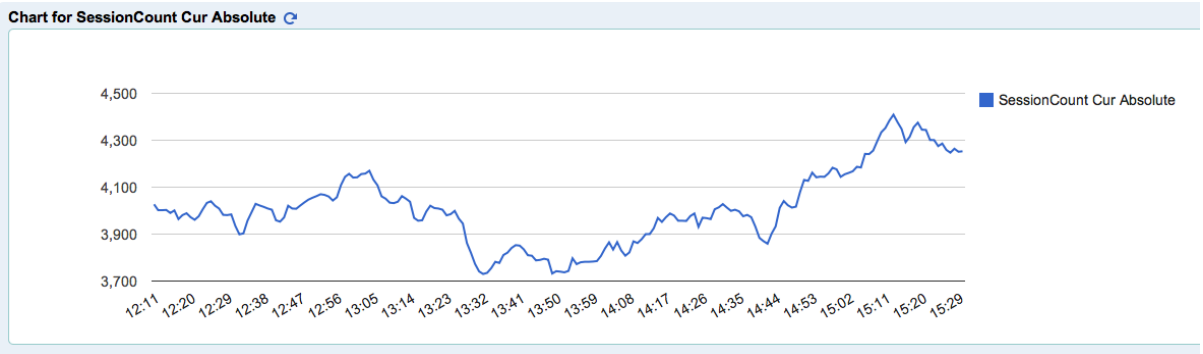
Name	Status	Value
SessionCount		3387
ThreadCount		837
OrderExportCounter-Failed-1h		none yet

Read more about thresholds.

Accumulate data and display it in charts

Sometime you might want to look closer at a monitored parameter and analyse its performance over time.

With **accumulators**, you may store MoSKito data for any needed period and display the accumulated info as a chart.



Read more about accumulators.

Make app's inner life visible

When you deal with a problem that a user faces, you have to understand which method or service failed. Such a translation from user's to dev's language takes time and is often quite painful.

Journeys make things way easier. They allow recording user actions in the form of actual calls/steps that occur inside a web application. Journeys show the methods/functions/services/resources a user triggers while using the app.

Read more about journeys.

Works with your app server

MoSKito's architecture allows it to work with any major application server. Tomcat, Jetty, JBoss, Glassfish, WebLogic... you may go on with the list.

Unless you're using something very specific and extraordinary, there's one less thing to worry about.

Use MoSKito on your mobile device

We have mobile versions for MoSKito-Essential (**MoSKito UI**) and MoSKito-Control (**MoSKito-Control App**) for iOS, Android apps are coming soon.

Having these on your mobile device simply means you know how your application is doing at any moment, wherever you are.

Other Features

- MoSKito-Inspect, embedded user interface for web applications.
- Duplicate call detection.
- Cumulated 'time-spend-in' component detection.
- Support for major logging frameworks.
- Built-in monitoring for threads, memory and system values.
- Thread-dump and Thread creation/deletion monitoring (ThreadHistory).
- Measurement in nanoseconds, presentation in different time units.
- Custom time intervals and user-triggered intervals.
- SQL queries monitoring.

Read Next:

MoSKito Concepts