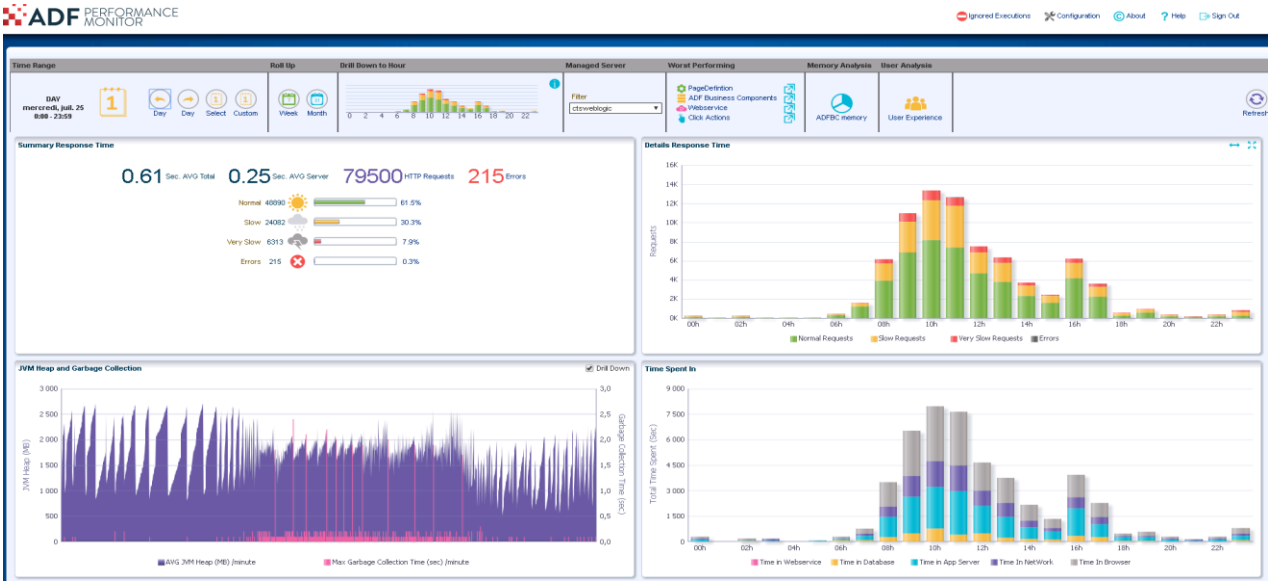
AMIS

ADF PERFORMANCE MONITOR

Whitepaper

Measuring, Analyzing, Tuning, and Controlling the Performance of Oracle ADF Applications



Author

Frank Houweling

Function

Principal Performance Consultant

ADF PERFORMANCE MONITOR

Contents

Executive Overview **3**
Introduction **3**
Oracle ADF applications and performance 3
ADF Performance Monitor 4
Features 5
Errors/Faults 6
HTTP Request and HTTP Response Network Time 7
Browser Load Time 8
ADF Click Actions 9
HTTP Request details and ADF call stack 10
Filter on User ID 10
ADF Call Stacks 11
Troubleshoot Slow SQL Queries 12
JVM performance 13
ADF BC Memory Analyzer 13
Warnings and suggested solutions 14
Week and month trend analysis with percentiles 15
Active End-Users and HTTP sessions 15
End-User Metrics 16
Worst performing executions in ADF BC and ADF Model layer 17
Support for EJB/JPA (EclipseLink) 18
Support for Soap / REST 18
Export to Excel or CSV File 18
Product architecture **19**
How it works 19
1. Metrics collection 19
2- Metrics logging 19
3- Dashboard reporting application 19
Monitored Events 19
Overhead 19
Clear error messages 19
Configuration **20**
Prerequisites 20
Configuration 20
Custom instrumentation 20
Turn on/off at all times 20
Installation guide 20
Training 20
Website 20
Use in JDeveloper during Development 21
Conclusion 22
Resources 22
Disclaimer 22
Appendix A: Comparison with other tools **23**

Executive Overview

The AMIS ADF Performance Monitor enables enterprises to maximize the value of ADF applications. Applications often contain frustrating blind spots and mysterious, recurring problems that are often difficult to identify, diagnose, and fix. With business-critical applications, it's very important to have a simple and fast way to monitor, diagnose, and resolve application problems before they affect revenue. The ADF Performance Monitor is specifically designed for measuring, analyzing, tuning, and controlling the performance of Oracle ADF applications. It helps enterprises by delivering insight into real end-user experiences. It helps development; QA teams and administrators detect, analyze and resolve common and less common issues in response times and resource usage of ADF applications. This document gives more information about the architecture, features and implementation of the ADF Performance Monitor. For more information please consult AMIS.

Introduction

Oracle ADF applications and performance

A good performance is the key to the success of a web application. Oracle ADF applications are no exception to this rule. Identifying and diagnosing bottlenecks in Oracle ADF applications can be time intensive, costly and quite a challenge. ADF is a powerful, advanced and highly configurable framework that is very performing and scalable if the ADF developer chooses the right combination of parameter settings. However, web applications in general and ADF applications have many pitfalls that that can be circumvented by choosing the correct performance configuration parameter settings. In most cases, unfortunately, the default values are not the most optimal values. Frequently even experienced ADF developers cannot pinpoint why an ADF application is slow. In this case information of what is happening behind the scenes is very useful in order to get a better understanding of their ADF application.

ADF Performance Monitor

The ADF Performance Monitor detects and identifies the causes of performance problems in a production, test and development environment. The tool consists of a library that is attached to the ADF application and a separate dashboard reporting application. The library collects runtime performance metrics. During the development phase issues are reported in Developers console logging. In a test or production environment, issues are reported real-time in the dashboard application (*Figure 1*).

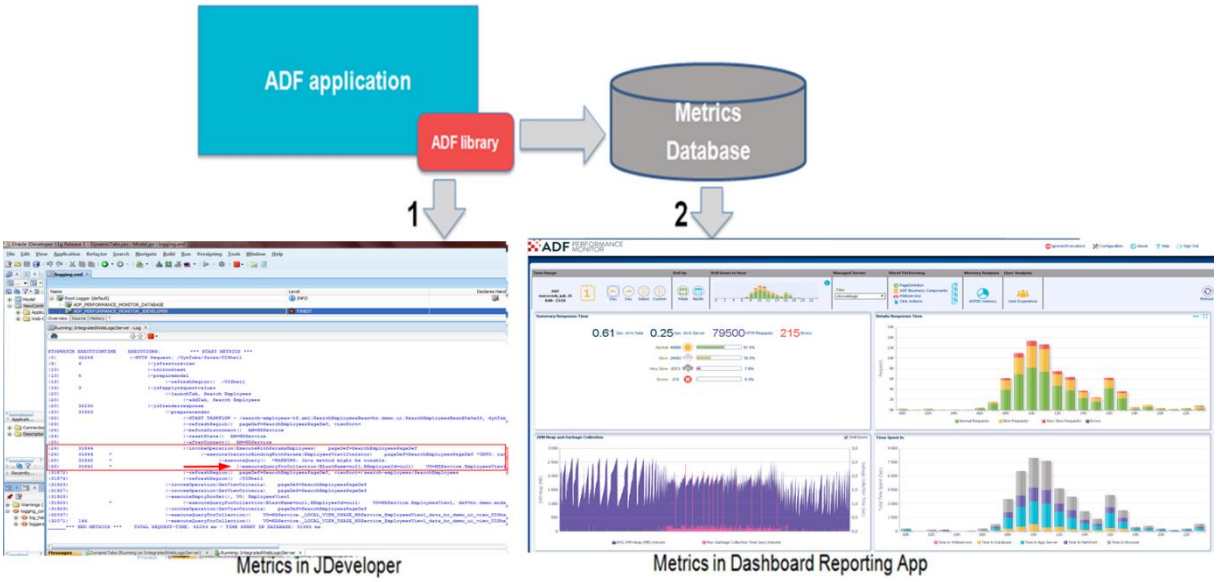


Figure 1: Performance metrics are reported in JDeveloper during development (1). In a test or production environment, issues are reported in a separate dashboard application (2).

With the ADF Performance Monitor development-, QA- and operation teams can:

- Get real-time and historic (today, yesterday, last week/month) overview of the performance.
- Get visibility/insight in real end-user experience
- Get visibility in which layer HTTP request process time is spent. The dashboard shows time spent in database, webservice, application server, and network/browser load time
- Get insight in what is happening inside the ADF application and in the ADF framework (what methods, operations and queries are executed, when and how often).
- Detect bottlenecks, inefficiencies and typical bad practices during the whole lifecycle of an ADF application. With this insight better application-architecture design decisions can be made.
- Make the ADF application more scalable (optimal utilization of infrastructure, hardware and licenses)
- Filter on WebLogic Managed Server, user ID, user session, business transaction, e.g. to troubleshoot problems
- Get insight in usage of ADF Business Components memory
- Get insight in errors/exceptions, their type, severity, ADF call stack and their stack traces (both technical and functional). And troubleshoot them quickly.

In Appendix A: Comparison with other tools - a comparison is made with other performance monitoring tools in the ADF landscape.

Features

The dashboard *(Figure 2)* gives a 24/7 overview of the performance in a selected time range. It shows real-time critical information about the ADF application's performance; are response times within or outside SLA boundaries? What is the error rate? What is the health of the JVM? Is immediate action required? The dashboard is separated into 4 regions:

1. Summary of real user experience HTTP request response times (top left). Categories:
    - Normal (well within the SLA range)
    - Slow (somewhat outside the SLA range)
    - Very slow (well outside SLA boundaries, considered seriously problematic)
    - Errors/Faults
2. Details of HTTP response times over time range, default the current day (top right). This chart makes visible when load is high (and how high), how the response times (very slow, slow and normal). When there is more red (and yellow) colored parts in the bar chart this should be a trigger to drill down to this hour for further analysis in order to find the root cause.
3. Layer where the HTTP request process time is spent (bottom right); time spent in the database, webservice, application server, network, and browser (bottom right)
4. JVM performance metrics (bottom left)

Figure 2: Performance overview in a selected time range (month, week, day, hour, 5 minutes).

Depending on the performance targets or SLA, QA and operation teams can configure how the monitor should interpret HTTP request response times. In the menu a time range can be selected *(Figure 3)* from a 5-minute level to a level of a month. To any time range can be drilled down, or rolled up (5-minute, hour, day, week, month), and navigate to next or previous time range.

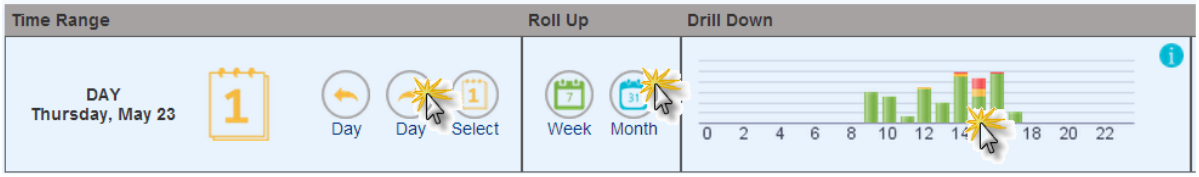


Figure 3: Drill down, roll up, or navigate to the next or previous time range.

Errors/Faults

On the top left in the dashboard main page, the number of errors over the selected time range *(Figure 4)* is shown. Insight in the number, type and severity of errors that happen in a test or production environment is crucial to resolve them, and to make a stable ADF application that is less error-prone. Application errors (and their stack traces) are often hard to retrieve or take a lot of time to find. Project teams commonly depend on the reports of end-users and testers, and they typically do not report all errors or provide insufficient information about them. Operational teams do not always have the time to monitor for errors in the WebLogic console or the Enterprise Manager or to wade through large log files to manually piece together what happened. The errors are collected by the ADF Performance Monitor to address this issue. Development, QA and operational teams can drill down to the error messages, their type, severity, and their stack traces to quickly troubleshoot errors and make the application more stable.

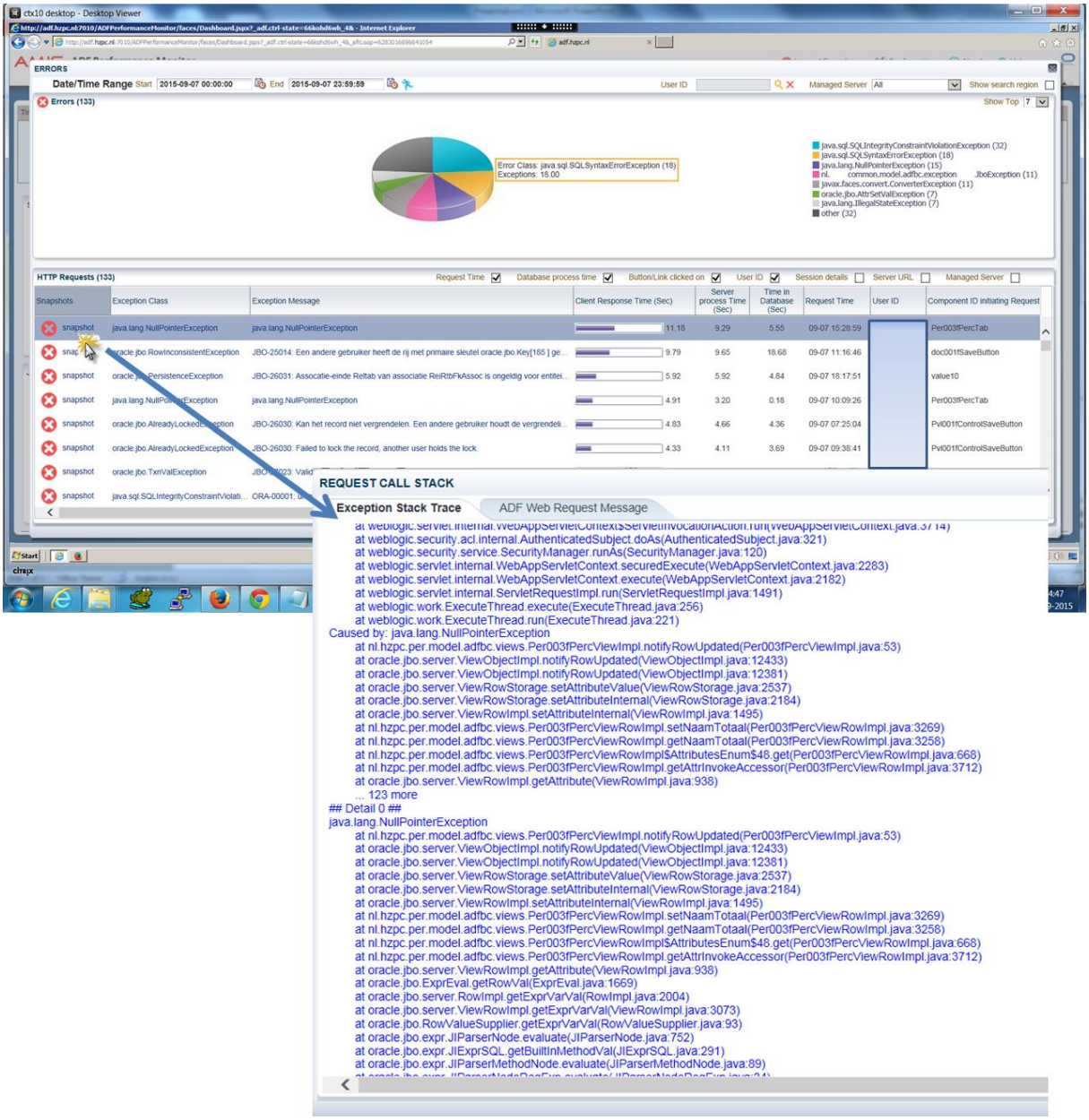
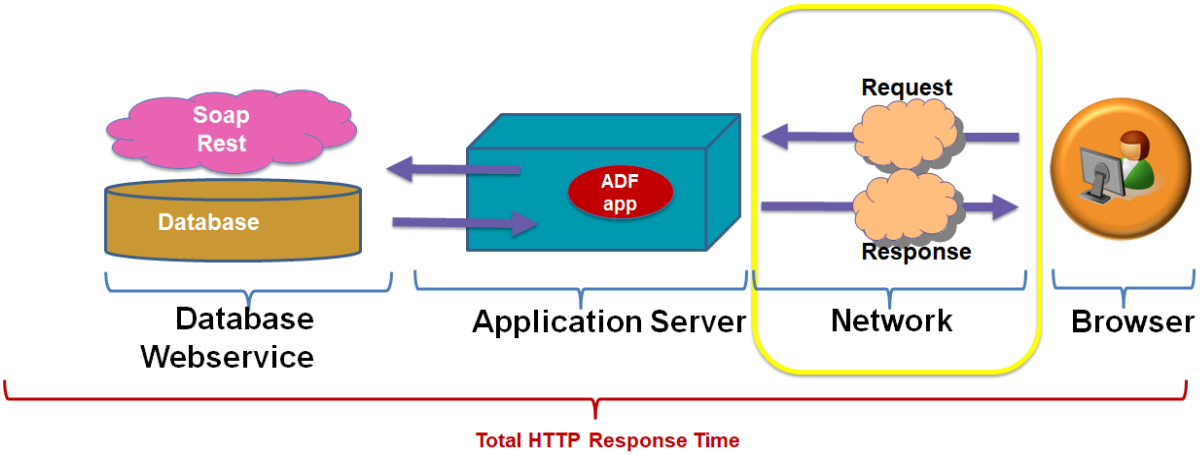


Figure 4: Top 10 Errors/Faults overview by Occurrences. Example of a *NullPointerException* exception stack trace.

HTTP Request and HTTP Response Network Time

Network time is the time that it takes to send a HTTP request from a browser (http request network time) to the application server and from the application server back to the browser (http response network time).

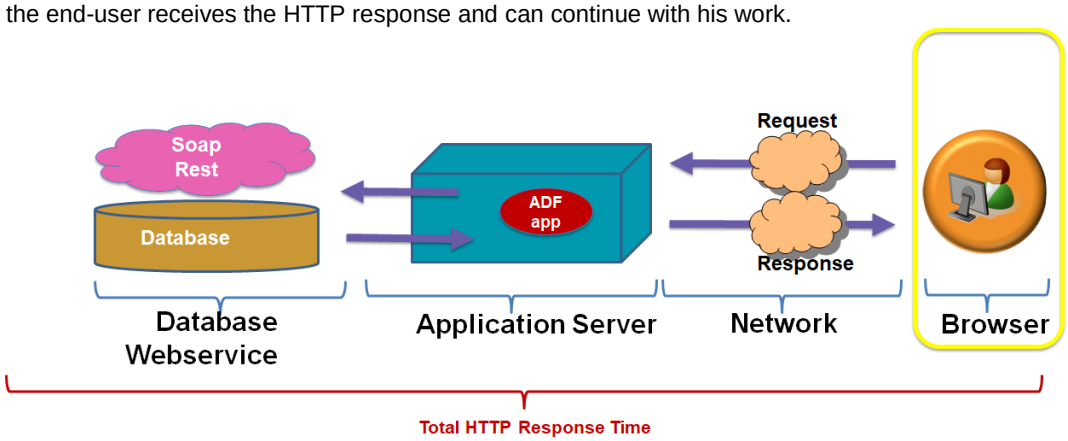


The ADF Performance Monitor shows for each time range (figure 5, right bottom) the layer where the time is spent. It shows the time spent in the *database*, *webservice*, *application server, network, and browser load time*:

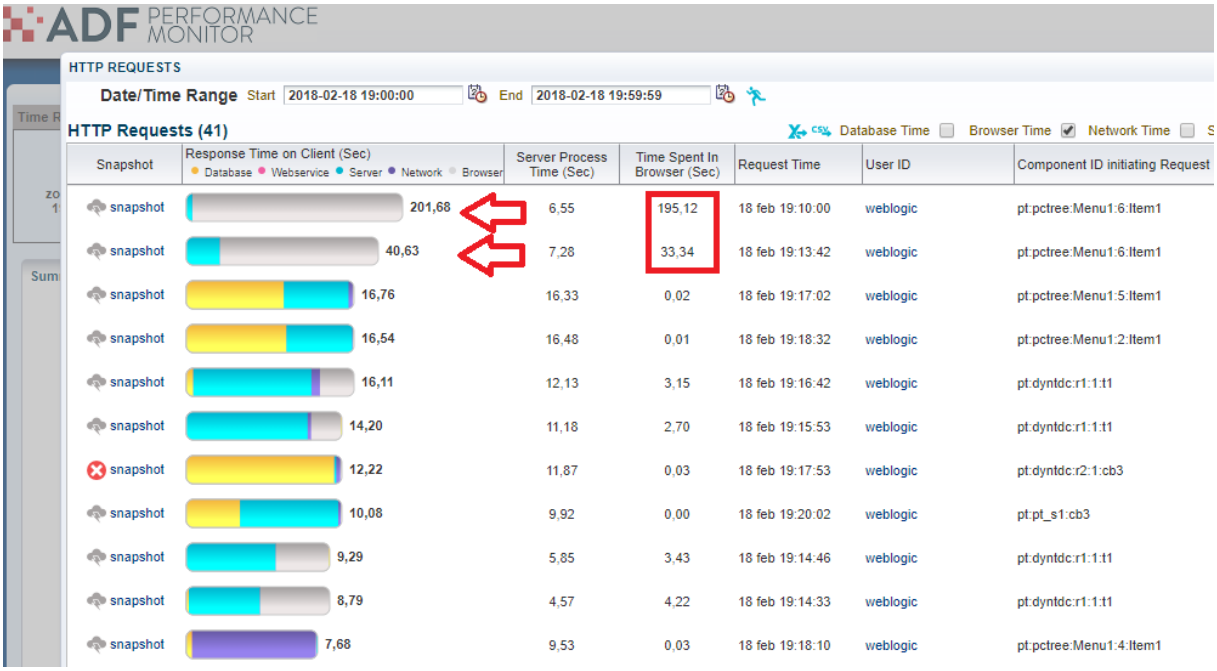
Figure 5: At the top right graph (minute overview of hour overview performance) we can see a lot of red in the bars. Specifically, in this hour from 11:00 to 12:00 – apparently there were many very slow requests. In the graph at the right bottom we can now explain what happened; there were big network problems (purple colour at right bottom graph).

Browser Load Time

While looking for performance problems at the usual suspects (server, database, webservices, network), the real problem can also be found in the often-overlooked browser load time. The browser lead-time is the time that a browser needs to build up the DOM tree and load the page. The browser load time is just an important part of the HTTP request and response handling as is the time spent in the application server, database, webservices and network. The browser load time can take a few seconds extra time on top of the server and network process time before the end-user receives the HTTP response and can continue with his work.



The browser load time can sometimes be exceptionally slow (see figure 6). Often this is caused by a 'bad' page design where too much ADF components are rendered and displayed at the same time; too many table columns and rows, but also too many other components can cause a slow browser load time.



Snapshot	Response Time on Client (Sec) (Database, Webservice, Server, Network, Browser)	Server Process Time (Sec)	Time Spent In Browser (Sec)	Request Time	User ID	Component ID initiating Request
snapshot	201,68	6,55	195,12	18 feb 19:10:00	weblogic	pt:pctree:Menu1:6:Item1
snapshot	40,63	7,28	33,34	18 feb 19:13:42	weblogic	pt:pctree:Menu1:6:Item1
snapshot	16,76	16,33	0,02	18 feb 19:17:02	weblogic	pt:pctree:Menu1:5:Item1
snapshot	16,54	16,48	0,01	18 feb 19:18:32	weblogic	pt:pctree:Menu1:2:Item1
snapshot	16,11	12,13	3,15	18 feb 19:16:42	weblogic	pt:dyntdc:r1:1:t1
snapshot	14,20	11,18	2,70	18 feb 19:15:53	weblogic	pt:dyntdc:r1:1:t1
snapshot	12,22	11,87	0,03	18 feb 19:17:53	weblogic	pt:dyntdc:r2:1:cb3
snapshot	10,08	9,92	0,00	18 feb 19:20:02	weblogic	pt:pt_s1:cb3
snapshot	9,29	5,85	3,43	18 feb 19:14:46	weblogic	pt:dyntdc:r1:1:t1
snapshot	8,79	4,57	4,22	18 feb 19:14:33	weblogic	pt:dyntdc:r1:1:t1
snapshot	7,68	9,53	0,03	18 feb 19:18:10	weblogic	pt:pctree:Menu1:4:Item1

Figure 6: HTTP Request overview – ordered by slowest HTTP requests. In the case the two slowest have an extreme high browser load time (grey colour). We see that the browser load time is 195 and 33 seconds.

ADF Click Actions

A click action is the start trigger event of an HTTP request by the browser, by an action that a user takes within the UI. These are most often physical clicks of end-users on UI elements such as buttons, links, icons, charts, and tabs. But it can also be scrolling and selection events on tables, rendering of charts, polling events, auto-submits of input fields and much more. With monitoring by click action you get insight in the click actions that have the worst performance, that cause most errors, that are used most frequently, e.g. You can see in which layer (database, webservice, application server, network, browser) the total execution time has been spent, ordered by total (sum) processing time. You can SLA monitor the business functions that are behind the click actions – from the perspective of the end-user.

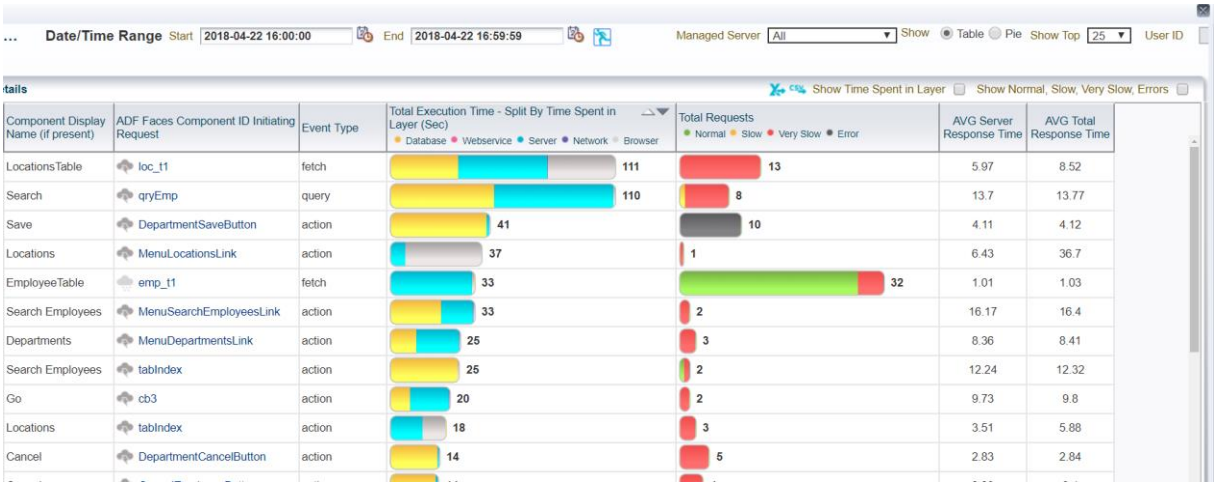


Figure 7: The component with the ID loc_t1 consumes the most sum execution time in total (1,8 minute, around 110 seconds). We see that the Java class component is an *oracle.adf.RichTable*, displayed as *LocationsTable*, and that the event is a fetch event. We see that 13 HTTP requests are very slow (red). And that almost one third of the request time is spent in the browser (grey). This should be a trigger to further investigate.

The overview shows:

- *Component ID* (ADF Faces component ID that started the request)
- *Component Type* (ADF Faces component Java Class)
- *Display name* (label/text if present on component)
- *Event type* (action, query, fetch, valueChange, selection, popupRemove, dvtImageView, e.g.)
- *Total* (sum) processing time (split by time spent in database, webservice, application server, network, browser)
- *AVG Server processing time*
- *AVG End-user time* (exactly as the end-user experiences it in the browser)
- *Total requests* (split by SLA; error rates, normal-, slow-, and very slow requests)

This overview shows on a high-level view what click action has the worst performance (is responsible for the most total processing time, and thus where we can win the most in terms of performance). For example, very slow poll action of type *oracle.adf.RichPoll*:

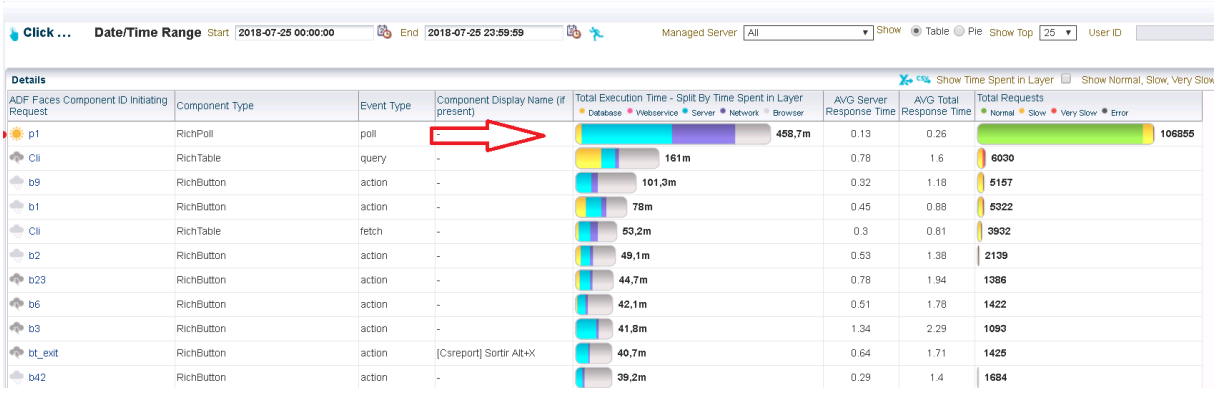


Figure 8: We see here that a poll event (ADF Faces component of type *oracle.adf.RichPoll*, with id 'p1') is responsible by far for the most total processing time (!). On this day there were in total 106.855 poll requests. That is more than one third of all the HTTP requests (296.435)!

HTTP Request details and ADF call stack

After clicking on the number of HTTP requests in the dashboard, a popup is shown with the HTTP request details sorted on client response time – ascending (the worst HTTP response times are on top – see figure 9).

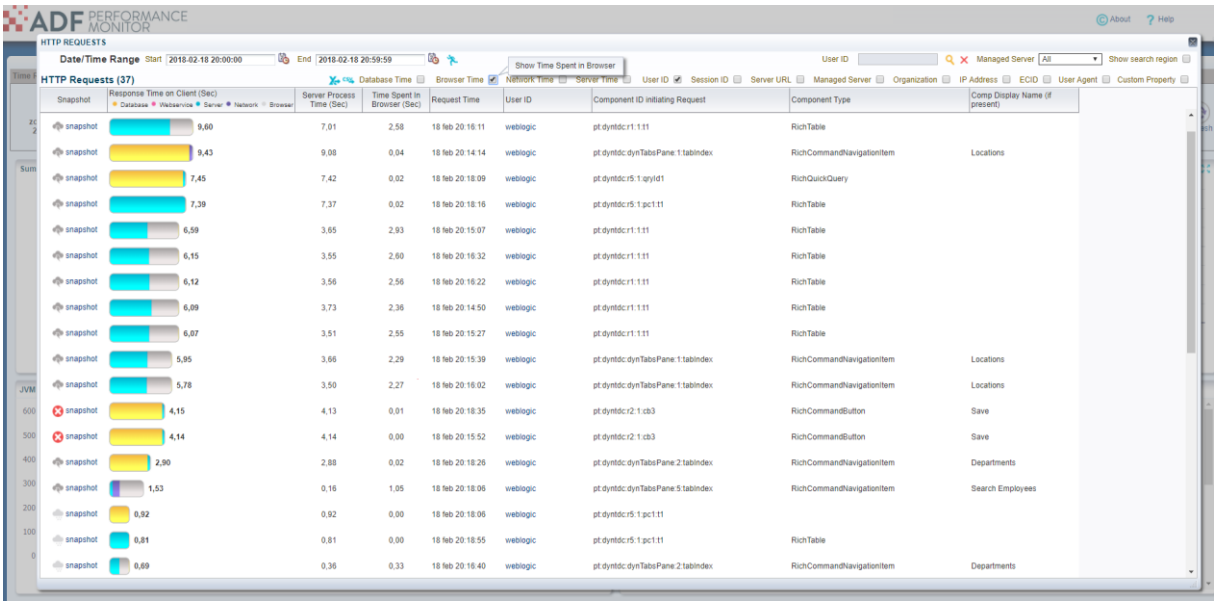


Figure 9: HTTP request details

The monitor shows HTTP requests details for further analysis:

- Total end user experience response time
  - Time spent in application server (blue)
  - Time spent in the database (yellow)
  - Time spent in webservice (pink)
  - Time spent in network (purple)
  - Time spent in browser (grey)
- Link to ADF call stack (snapshot)
- User ID
- Timestamp of Request
- Component initiating request (Click action Id, ADF component type, component display name)
- Application server URIs
- WebLogic Managed Server
- Session details (link) to zoom in to all HTTP requests of this session
- ECID
- User agent
- Custom Property
- User Organization

Filter on User ID

In case end-users (or local testers) are complaining about todays or yesterday's performance, the monitor helps to quickly troubleshoot. With the monitor you can filter on a specific end-user ID and analyze/troubleshoot the performance for this specific user ID. In addition to this there are also overviews of the top end-users and end-users with the worst performance. The true picture for every individual user is visible. Every user interaction (like clicks and scrolls) is logged. You can trace every session of your end-user's experience.

ADF Call Stacks

A *call stack* (*Figure 9*) gives visibility into which ADF method caused other methods to execute, organized by the sequence of their execution. Including runtime method and query parameter values. This is the same call stack that is printed in JDeveloper during development. A complete breakdown of the HTTP request is shown by actions in the ADF framework (Fusion lifecycle phases, model (BindingContainer) and ADF BC executions, start & end of taskflows, e.g.), with elapsed times and a view of what happened when. The parts of the ADF Request that consume a lot of time are highlighted and indicated with an alert signal.

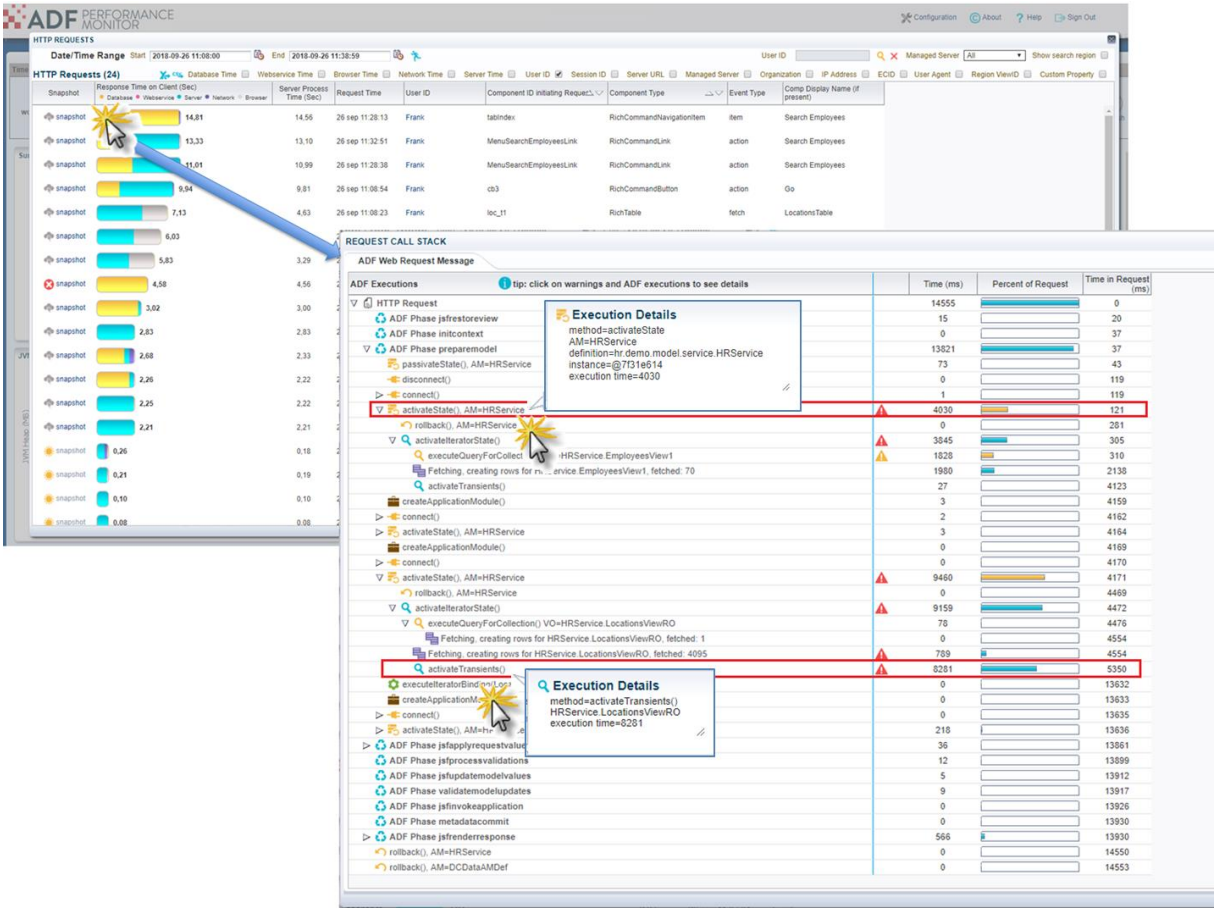


Figure 9 : We can see in the callstack a slow activation of the ApplicationModule called HRService (4030 milliseconds). Later in the HTTP callstack we see a slower activation of the different HRService instance (9460 milliseconds), caused by a slow activation of transient attributes in activateTransients() of ViewObject HRService.LocationViewRO (8281 milliseconds). We can see that there are 4095 rows of LocationViewRO fetched (!)

When an execution is clicked, a popup comes up and shows detail information like ViewObject usage name, ViewObject definition name, bind variable values, applied ViewCriteria names, number of rows fetched from database, ADF object names, e.g.

Troubleshoot Slow SQL Queries

In the Configuration settings of the ADF Performance Monitor, a parameter can be set whether the monitor should log the SQL and DML statements of ADF Business Components or EJB/JPA or not. If yes, a threshold can be set when it should be collected. For example: all SQL Queries and SQL DML statements over five seconds (5000 milliseconds). Now runtime SQL executed against the database can be analyzed (Figure 10):

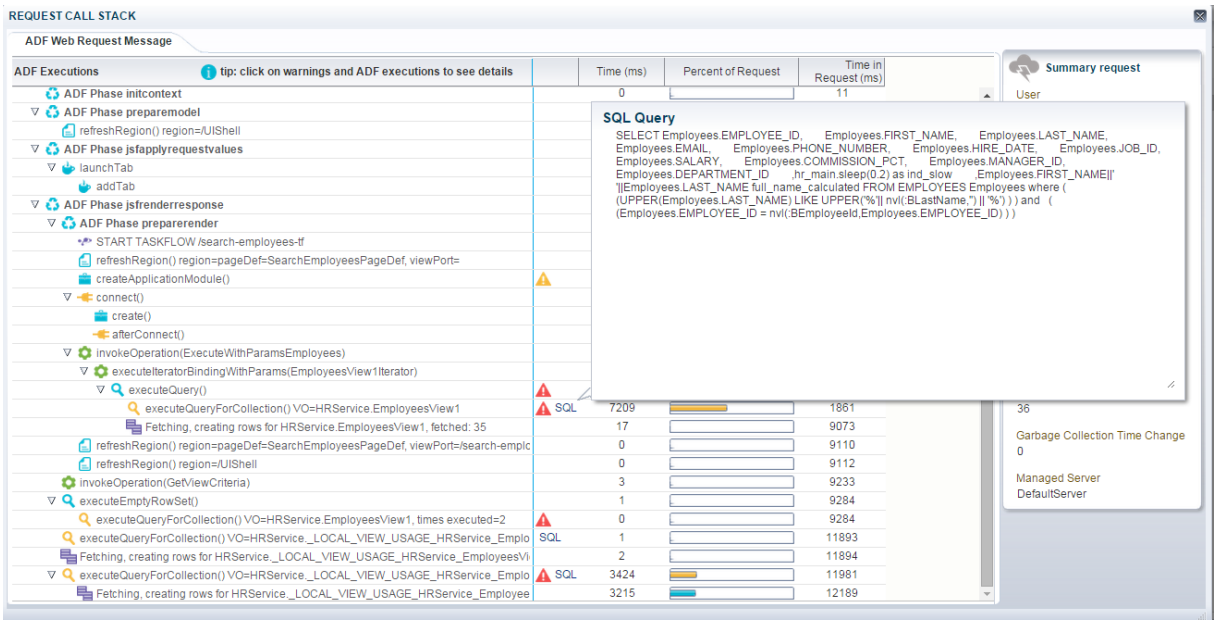


Figure 10: ViewObject SQL executed runtime against the database. The runtime bind parameter values are also visible by clicking on the ViewObject *executeQueryForCollection()* method.

JVM performance

An important aspect of performance management is a healthy JVM. The ADF Performance Monitor shows real-time or historic heap usage and garbage collection times *(Figure 11)*. If garbage collections are running longer than a configurable threshold (for example 20 seconds) a warning sign is shown. This is often an indication of a problem – such as a freeze of all current requests because the JVM cannot clear enough memory.

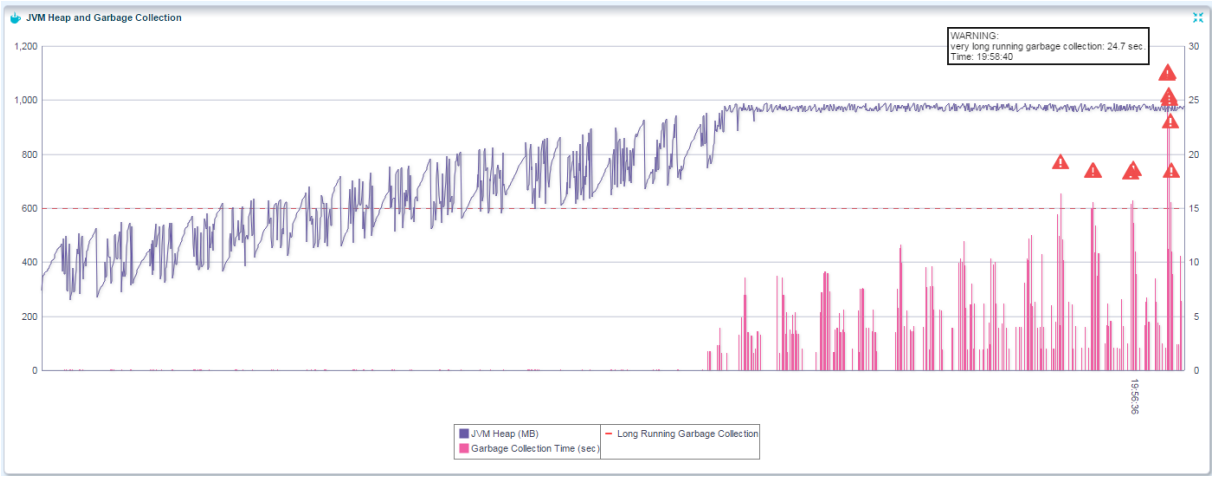


Figure 11: Example of very long running garbage collections (pink line). The heap space (purple) over time evolves more into a horizontal line than the saw-tooth shaped line a healthy JVM would be characterized by. In this case an out-of-memory-error occurred, and the server needs to be restarted. This should be a trigger to investigate whether the JVM heap space is set too low or that the ADF application overconsumes memory. The request call stacks should be analysed in order to find the root cause of the memory overconsumption. This typically happens when the ADF application fetches too many (database) rows and too many attributes.

ADF BC Memory Analyzer

Like other web applications, ADF applications potentially use a lot of memory. Many times, the root cause of a high memory usage is that application data retrieved from the database into memory is not properly limited; too many rows with too many attributes are fetched and held in memory. The ADF BC Memory Analyzer (Figure 12) detects how many database rows are fetched by ADF ViewObjects. Loading too many database rows can lead to memory overconsumption. When a ViewObject loads more than 250 rows (configurable) into memory the monitor gives a warning sign in the call stacks. It also suggests solutions - like using ViewObject range paging or setting an appropriate (maximum) fetchsize on the ViewObject.

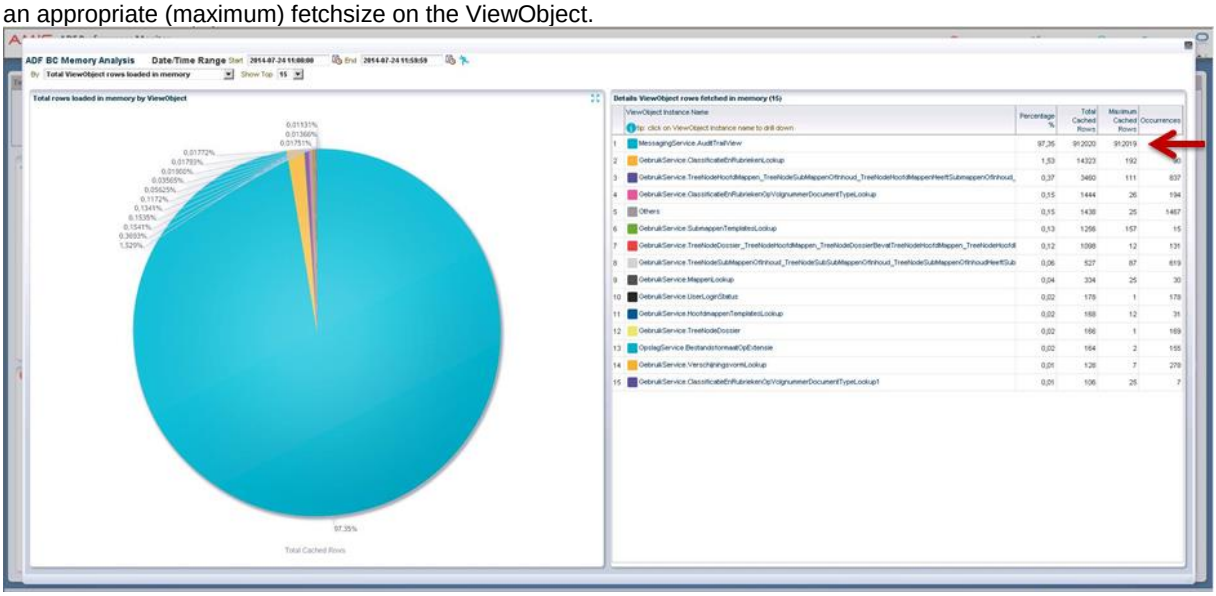


Figure 12: ADF BC Memory Analyzer. In this production ADF app, for a single ViewObject instance, more than 900.000 rows were loaded in the ADF app (blue)!

Warnings and suggested solutions

During the whole application lifecycle, QA teams can make sure problems are found and dealt with. They can routinely check to ensure the application meets the desired quality. In the callstacks clickable warnings signs (*Figure 13*) are shown for slow executions and detected inefficiencies. These give a quick help with suggested solution(s) and a link to more detailed help.

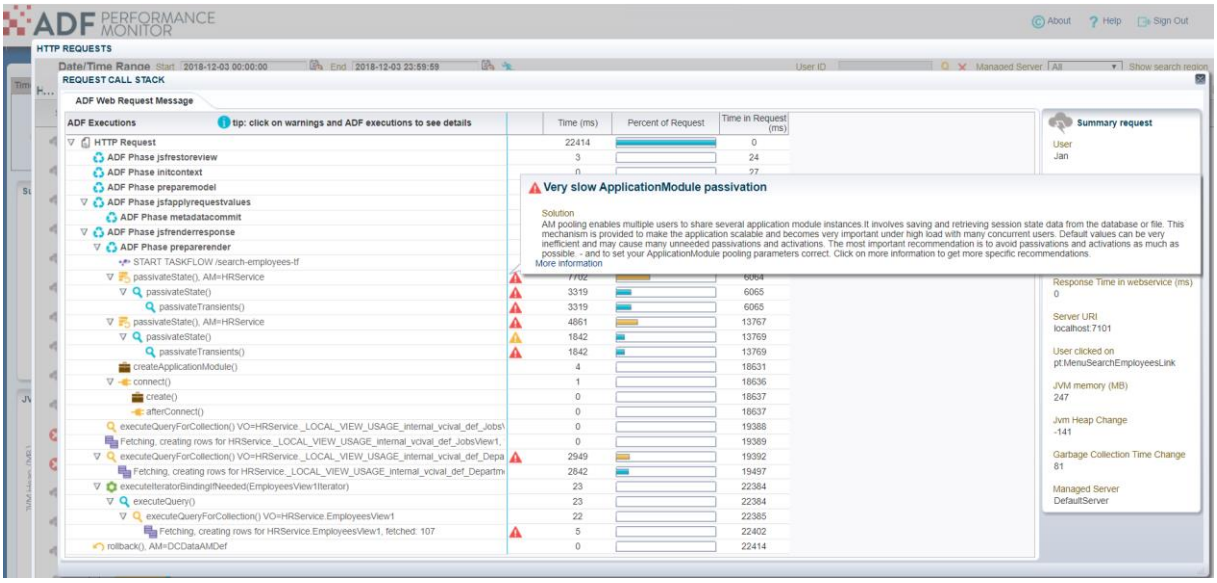


Figure 13: Very slow ApplicationModule passivation. By clicking on the warning image, a quick help popup is shown with a suggested solution. More detailed help is also available.

Warnings and suggested solutions are shown for example in the following cases:

- Slow executions
    - *ViewObject queries*
    - *EntityObject DML operations*
    - *ApplicationModule activations, passivations, transactions*
    - *PL/SQL calls*
    - *BindingContainer operations, iterator executions*
    - *Java methods*
    - *Webservice calls*
- Slow passivation and activation of transient ViewObject attributes
- Fetching a very high number of rows from database into Java memory
- Multiple redundant ViewObject query executions during the same HTTP request
- Inefficient ViewObject fetchsize
- Inefficient BindingContainer iterator rangesize

Week and month trend analysis with percentiles

Percentiles, when compared with averages, tell us how consistent our application response times are. Percentiles make good approximations and can be used for trend analysis, SLA agreement monitoring and daily to evaluate/troubleshoot the performance. For example, see figure 14; after a new release on 17 June, the 95, 90th, and 50th percentile response time decreased – the performance is improved.

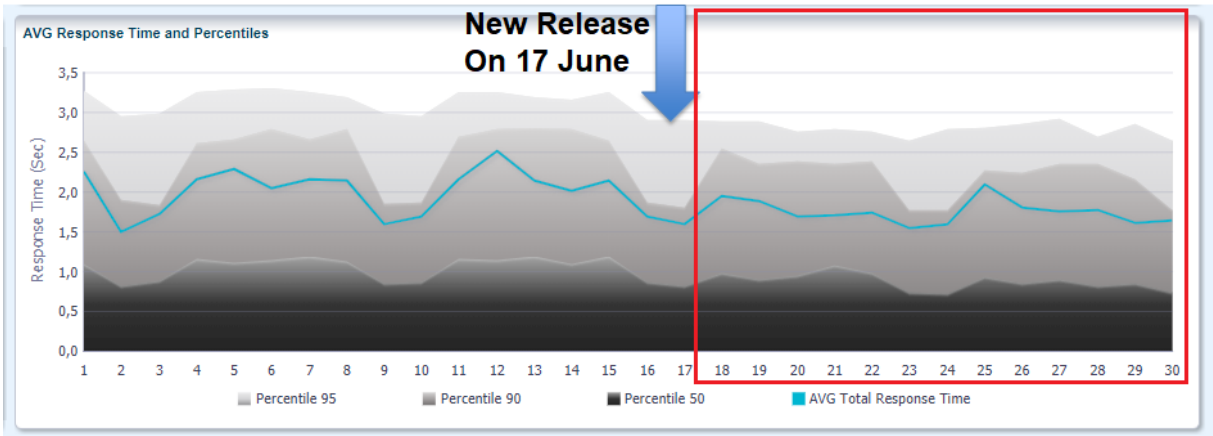


Figure 14: After a new release on 17 June, the 95, 90th, and 50th percentile response time decreased – the performance is improved.

We can use percentiles for all kind of performance evaluations. For regressions and trend analysis after new releases. Did we really improve the performance or not? Sometimes the performance increases or decreases after new releases – it would be useful if we the visibility to see and recognize this.

Active End-Users and HTTP sessions

Active End-Users and active HTTP sessions that really click around are shown in this overview – very useful to evaluate number of end-users and sessions that are active (figure 15). On one managed server – or on all managed servers together. You can compare the load for each server and compare the values to all the other metrics in the ADF Performance Monitor like JVM, percentiles, SLA agreement metrics, time spent in layers, e.g.

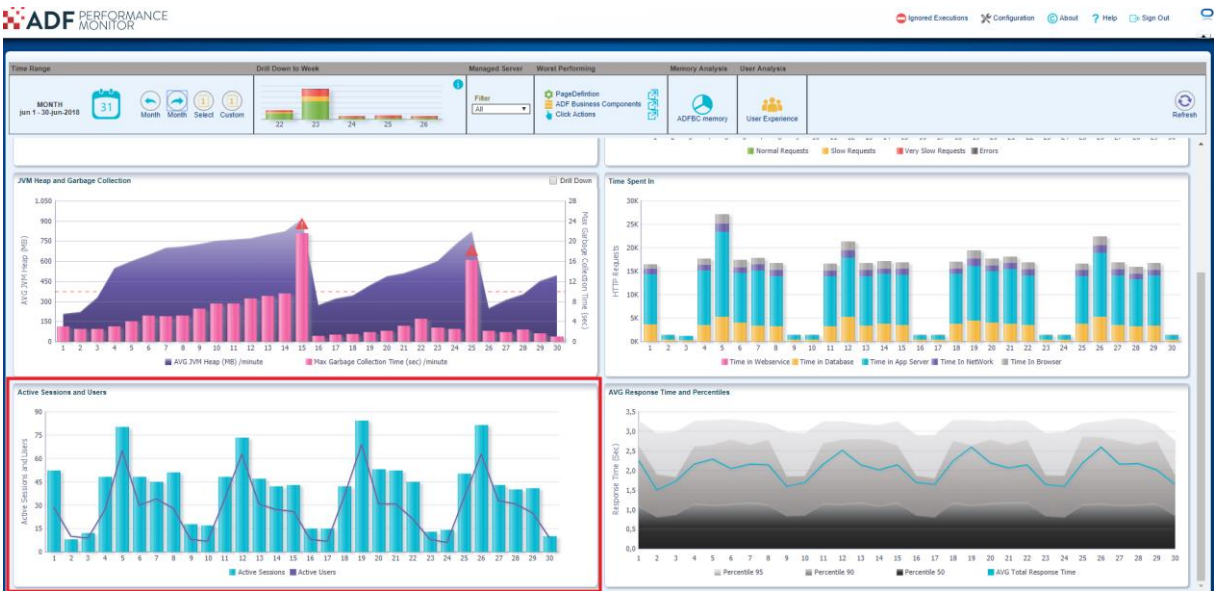


Figure 15: Active end-users and active HTTP sessions

End-User Metrics

There are several overviews of summarized end-user metrics. Like Worst Performance by User, Top Users by Requests, and Most Errors by Users (Figure 12).

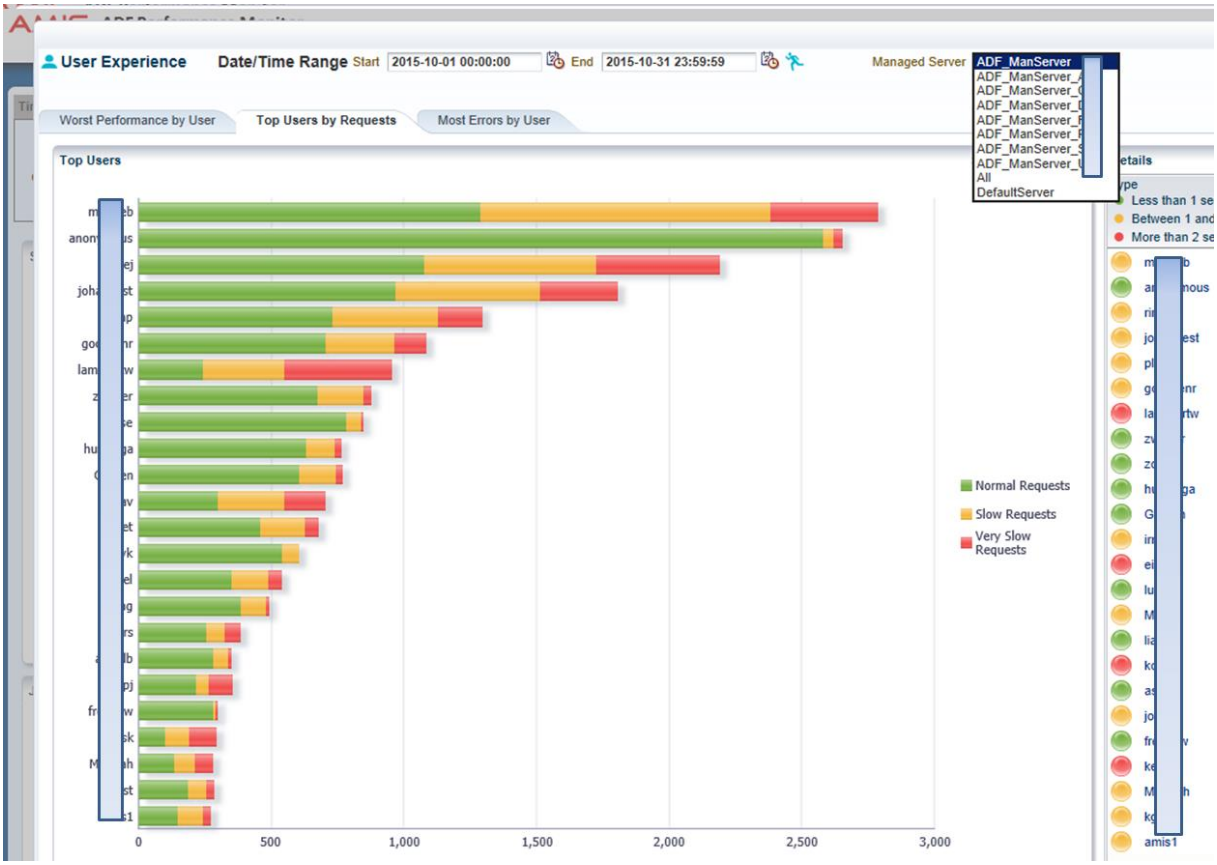


Figure 12: There are several overviews of summarized end-user metrics. Like Worst Performance by User, Top Users by Requests, and Most Errors by Users.

Worst performing executions in ADF BC and ADF Model layer

In various overviews the worst performing parts of the ADF BC and ADF Model layer are indicated. In these overviews bottlenecks can be found. The development team can act to improve the application. There are overviews for the worst performing executions of:

- ADF Business Components
- (PageDefinition) BindingContainer executions
- Webservices (calls to JAX-WS webservices)
- Java methods instrumented by the ADF developer

Worst ADF BC executions

Figure 13 is an example of an ADF BC overview. It shows very slow, slow and normal ADF BC executions:

- ApplicationModule pooling passivations / activations
- ApplicationModule transactions
- ViewObject queries
- EntityObject DML operations
- PL/SQL procedure/function calls executed from ApplicationModules

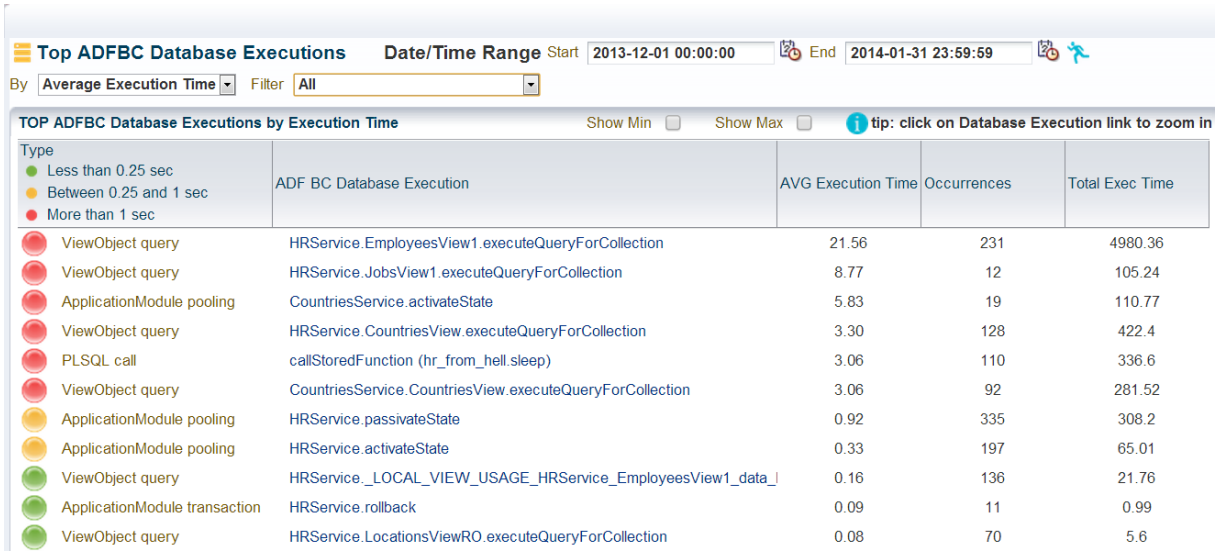


Figure 13: Overview of ADF Business Components by AVG execution time. First a marker (red, orange, and green) is shown (configurable). Then the type, the usage (instance) name, the AVG execution time, occurrences and the total time. The occurrences and their callstacks can be further analysed in a popup window.

ApplicationModule Pooling analysis

It is important to choose the right combination of ApplicationModule pooling parameter settings to make the application scalable. Not setting these parameter settings correct (and leaving them at the default values) can be very inefficient and may cause many unneeded passivations and activations. In the ADF BC overview (filtered on ApplicationModule pooling) the ApplicationModule pooling performance can be analyzed. This overview gives insight in the effect of these parameter settings (how often passivations and activations happen, how long their AVG and total execution time is). QA teams can research and experiment with these parameter settings on test environments during load tests and evaluate the performance results in the monitor. The monitor also detects which VO data are being activated and passivated. The root cause of long running activations/passivations is often unnecessarily passivated calculated and transient attributes in combination with an unnecessarily high number of fetched database rows.

Worst BindingContainer executions
The monitor shows very slow, slow and normal BindingContainer executions:

- Operations (all standard ADF operations, client methods on ApplicationModule and ViewObject)
- Iterators from the executable section

Total execution time of worst performing executions
The performance impact of frequent invoked executions can be much higher than executions that occur only a few times but are very slow on average. For example, a 'slow' ViewObject query with an average time of 1500 milliseconds but executed 10k a day, has more impact than a 'very slow' ViewObject query with an average time of 15 seconds that is executed only 2 times a day. There are additional overviews for the total execution time.

Ignore well-known long running executions
Not all executions can be as fast as we would like them to be. For example, some SQL queries just need time to execute because it needs to perform a lot of processing. The ADF Performance Monitor has a feature to filter these identified executions from the overviews. A threshold can be set to ignore or show an execution.

Support for EJB/JPA (EclipseLink)

Not all Oracle ADF projects are built with ADF Business Components. Some organizations use EJB/JPA (EclipseLink) instead of ADF BC. The ADF Performance Monitor supports EJB/JPA. Amongst others EJB/JPA DataControl operations/methods, runtime invoked methods on EJB SessionBeans and SQL queries/DML actions executed are instrumented and visible on the ADF call stacks.

Support for Soap / REST

Webservices are supported. All webservice calls are instrumented and visible on the ADF call stacks. Time spent in webservice calls is shown on the main dashboard page.

Export to Excel or CSV File

You can export all overviews of the monitor to an Excel or CSV file. In this way you can save it for analysis later or do trend analysis after new releases, check whether the performance has been improved or not.

Product architecture

How it works

The product consists of an ADF library and a dashboard reporting application. The ADF library (attached to your application) collects runtime behavior and performance metrics. The metrics are saved in a separate database schema. The dashboard reporting application retrieves the metrics from the same database (*Figure 14*).

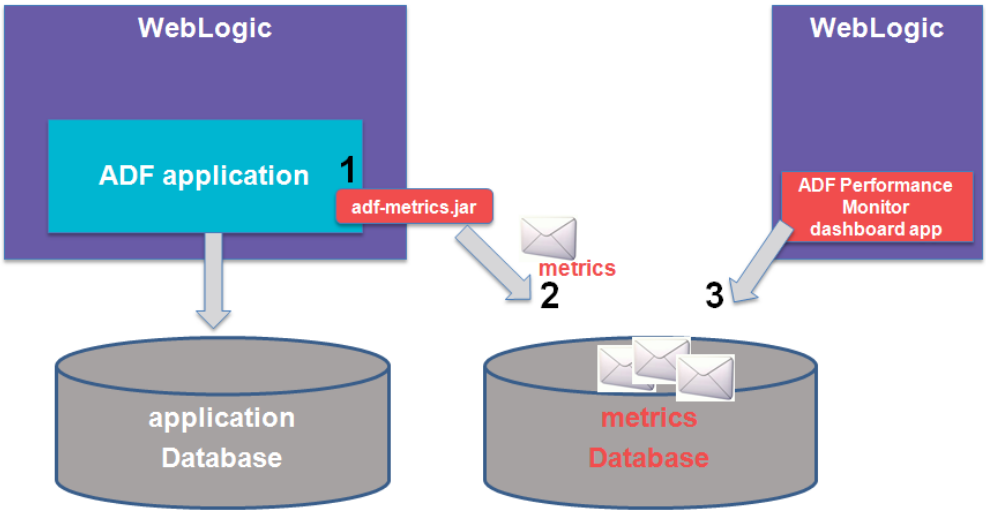


Figure 14: Architectural overview of the ADF Performance Monitor

1. Metrics collection

Raw metrics data is collected from every HTTP request by the attached ADF library. When the HTTP request is finished, the metrics data are buffered.

2- Metrics logging

A background job saves the metrics into an Oracle database. This database is a separate, different one than the database of your application - and preferably on a different machine to not impact machine resources of application database.

3- Dashboard reporting application

The dashboard reporting application is a web-based ADF application that can be accessed from any supported browser. This application is deployed on a separate WebLogic managed server and communicates with the metrics database.

Monitored Events

The monitor collects performance information from specific ADF objects.

Overhead

Load tests show that the overhead is less than 3 percent. This overhead is caused by the very detailed collection of performance metrics. The metrics collection can always be dynamically turned on and off (at runtime) . When the monitor is turned off there is no performance overhead because the metrics classes are not active.

Clear error messages

The ADF Performance Monitor logs clear error messages in the Enterprise Manager in case resources needed for the monitor are not available (for example if the metrics database is offline). Or in case an error occurs. This will not cause any application error in the ADF application.

Configuration

Prerequisites

The ADF Performance Monitor can be implemented on all ADF10g, ADF11g and all ADF12c applications. Any Oracle11g (also the free XE) database can be used for the metrics database. If ADF BC is used, the use of ADF BC framework extension classes is required.

Configuration

- An ADF library needs to be added to your ADF application. A very small amount of code needs to be added to instrument the ADF application. The metrics classes make use of extension points in the ADF framework in order to measure the time every action/step takes in an http request.
- On a test or production environment, a JDBC datasource must be configured on the managed server that connects to the metrics database.
- The dashboard application EAR file must be deployed on a separate WebLogic server instance to not impact WebLogic resources. This EAR can also be run locally in JDeveloper. Other application servers like Glassfish are not (yet) supported because ADF security is used.

Custom instrumentation

It is possible to instrument (calls to) custom Java methods and 3rd party libraries. Custom methods are visible on the call stacks and the Java methods performing executions overview.

Turn on/off at all times

The ADF Performance Monitor makes use of the standard Oracle ADFLogger and can be turned on and off at runtime at all times in the Enterprise Manager.

Installation guide

All configuration steps are well documented in the installation guide. The implementation takes less than 1 day, usually no more than a few hours. A user manual is available from within the dashboard application.

Training

AMIS can provide consulting and training on the ADF Performance Monitor, ADF performance tuning and ADF best practices (on location or online via webcast).

Website

All information concerning this product is available on our website – adfpm.com.

Use in JDeveloper during Development

The ADF callstacks of the ADF Performance Monitor can already be used during development. This enables developers to diagnose and solve performance problems at an early stage and to build an efficient, responsive ADF application before it goes into production. In JDeveloper, ADF developers can switch the metrics logging on and off with the standard ADFLogger of Oracle. For every HTTP request a call stack is printed. A call stack provides visibility into which ADF methods caused other methods to be executed, organized by the sequence of their execution.

A complete breakdown of the ADF request processing, including all ADF method executions, along with elapsed times is printed, organized by lifecycle phase. Now Inefficiencies can be identified from this report, like a long running ViewObject query of 31858 Milliseconds *(Figure 15)*:

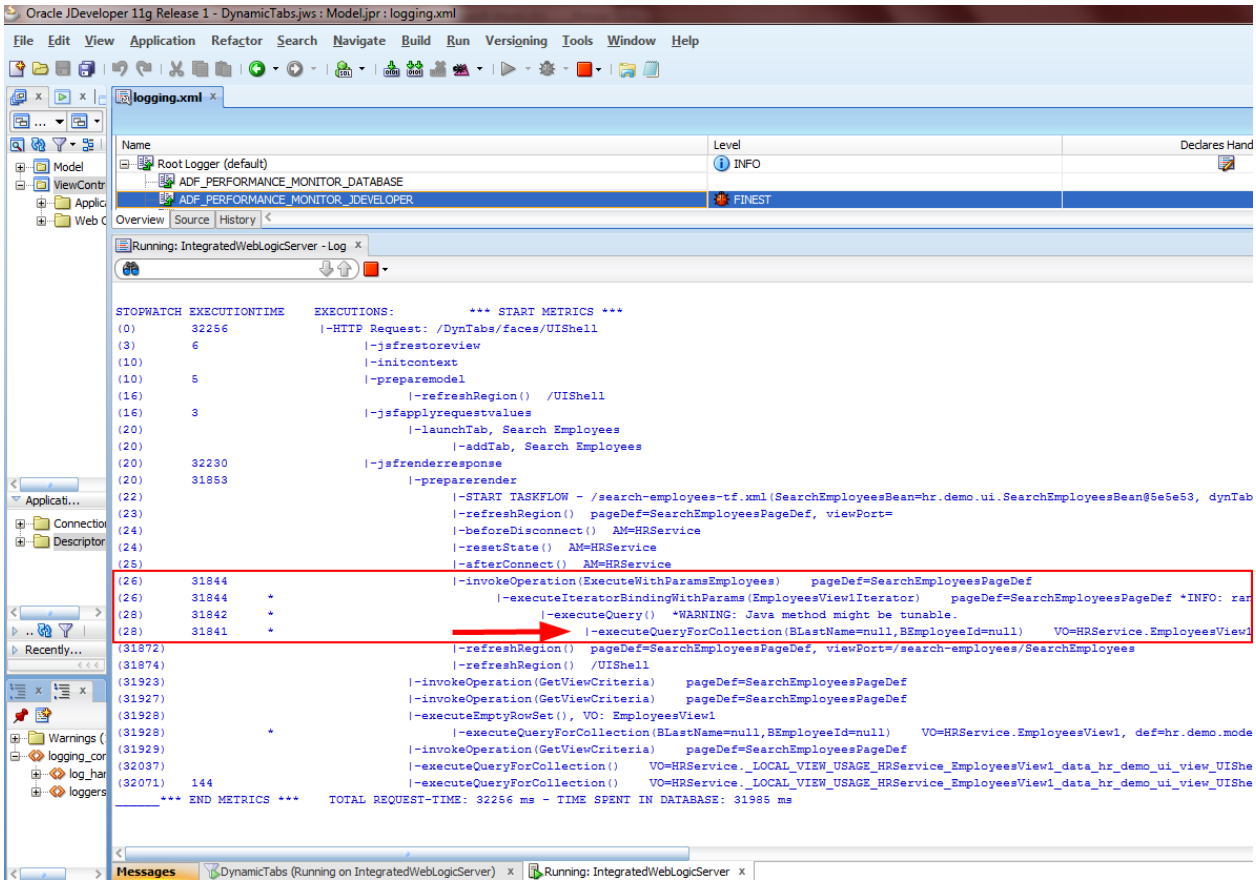


Figure 15: ADF metrics are printed in JDeveloper WebLogic Server Log (can be switched on and off). Shown is an example of a long running ViewObject query of 31858 milliseconds (usage name of HRService.EmployeesView1)

Conclusion

The ADF Performance Monitor is a tool for measuring, analyzing, tuning, and checking the performance of Oracle ADF applications. Throughout the whole application lifecycle, the development, QA and operation teams get insight into what is happening inside their ADF application. With this insight ADF developers can diagnose and solve performance problems already in an early stage, make better (architecture) decisions and can build more responsive and scalable ADF applications. With the warnings and suggested solutions they can circumvent common and uncommon performance problems, use best practices and deliver a higher quality. This will lead to an application that is more consistent and better to maintain. They can reduce the utilization of infrastructure, hardware and licenses.

Because of the performance overhead of only 3 percent, the ADF PM can easily be used in production environments. In this way performance bottlenecks and errors can be pro-actively monitored and solved.

Resources

More information on www.amis.nl/ADFperformancemonitor, including several demo videos.

Disclaimer

The information in this whitepaper is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality of this product products remains at the sole discretion of AMIS.

Appendix A: Comparison with other tools

Application performance tools on the market are generally not aware of the intricacies of the ADF framework. They focus on generic, low level JVM metrics or rather on high level business transactions (like 'add to cart' or 'pay bill'), but miss the relevant key actions and resource usage metrics of the ADF framework.
Oracle provides some support for performance analysis with WebLogic Enterprise Manager. Enterprise Manager provides only basic monitoring information for ADF applications. Also in JDeveloper there is some support for performance analysis through the ODL Analyzer. This tool can be useful in the development stage, but can't be used in test and product environments because of the amount of logging generated and the significant performance overhead.

In table below a comparison is made between different monitoring tools in the ADF landscape:

Monitoring Features	**WebLogic Enterprise Manager**	**Oracle Real User Experience Insight (RUEI)**	**Generic APM Tools (New Relic, AppDynamics)**	**ADF Performance Monitor**
High Level HTTP Requests Overview	±	+	+	+
Detailed HTTP Requests Diagnostics	-	+	-	+
Filter on User ID (Select User Sessions of interest, current or in history)	-	+	-	+
Insight in Real End User Experience	-	+	+	+
Time Spent in Database, Application Server, Webservice ,Network/Browser Load Time	-	-	+	+
JVM Metrics	+	-	+	+
Error Diagnostics	+	±	+	+
ADF Request Call Stack (Code Level Visibility of ADF Key Actions)	-	-	-	+
ADF Business Components Memory Analyzer	-	-	-	+
ADF Specific Warnings & Suggested Solutions	-	-	-	+
Worst ADF BC Slow SQL Queries (Runtime SQL Query and Bind Variables)	-	-	±	+
Worst ADF BC ApplicationModule Pooling (Passivations & Activations)	-	-	-	+
Worst ADF BC EntityObject DML operations	-	-	-	+
Worst PL/SQL procedure/function Calls			±	+
Worst (PageDefinition) BindingContainer Executions	-	-	-	+
Support for EJB/JPA (EclipseLink)	-	-	±	+
Support for JAX-WS / Apache CXF Webservices	-	-	-	+
Diagnostics & Warnings in JDeveloper During Development	-	-	-	+
Performance Overhead	0%	0%	2%	2%