Oracle® Cloud

Using Oracle Application Performance Monitoring

E60699-46
February 2021

Oracle Cloud Using Oracle Application Performance Monitoring,

E60699-46

Primary Author: Oracle Corporation

Contents

Preface

Oracle Application Performance Monitoring provides a platform for monitoring and managing your applications.

Topics:

Audience

Using Oracle Application Performance Monitoring is intended for users who want to monitor the performance of their applications.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Resources

For more information, see these Oracle resources:

- Oracle Cloud

  `http://cloud.oracle.com`
- Installing and Configuring Oracle Application Performance Monitoring
- Using Oracle Log Analytics

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introduction to Oracle Application Performance Monitoring

Oracle Application Performance Monitoring provides a platform for monitoring and managing your web applications.

Topics

About Oracle Application Performance Monitoring

Today's e-businesses depend heavily on their applications to allow critical business processes to be performed online. As more emphasis is placed on accessing information quickly, remotely, and accurately, you should take proactive steps to ensure that your online customers can successfully complete a transaction. Oracle Application Performance Monitoring (APM) is a cloud service that provides deep visibility into the performance of your application.

With Oracle Application Performance Monitoring, you can:

- Rapidly isolate application performance issues
- Drill down to related logs in context of a problem and find its root cause
- Gain end-to-end visibility into the performance of your application across all tiers
- Monitor end-user experience

Oracle Application Performance Monitoring is designed to provide insight into the application's performance, down to the operation and method level, as well as application requests across servers. Also, APM provides direct visibility into the production performance of applications, in shared environments. With the Oracle Log Analytics service integration, users can examine application logs seamlessly in the context of various application problems. Additionally, using the Oracle Infrastructure Monitoring service, administrators can get additional finer-grained monitoring data, such as specific page data or Ajax call data as well as per Service reporting metrics, which can help troubleshoot and optimize applications performance next to discovering the applications infrastructure.

Roles in Oracle Application Performance Monitoring

When your Oracle Application Performance Monitoring service instance is created, the following roles are provisioned by default:

Role	Tasks
Oracle Management Cloud Administrator	• Set up Oracle Application Performance Monitoring by deploying APM Java Agents on target hosts • Manage APM Java Agents • Configure alert rules
Oracle Management Cloud User	• View and monitor application performance • Drill down to related logs • View alerts

Read This Before You Begin

Here are some of the common terms and basic concepts regarding Oracle Application Performance Monitoring.

Asynchronous JavaScript and XML (AJAX) is a group of Web technologies used to implement asynchronous Web applications that communicate with a server in the background, without interfering with the current state of pages.

The **APM Java Agent** is a lightweight agent, which runs in the Java Virtual Machine (JVM) of a web application and collects performance monitoring data for Java web applications running in your data center or in the cloud.

An **application server** is a server on which applications and services are installed, hosted and operated. It is part of the middle-tier in a three tier architecture.

An **application request** is typically an HTTP request sent by the client application to the server.

A **server request** is an application server request that can come via HTTP or some other service. A server request might be called by an HTML page, an AJAX request, or another server request.

Garbage collection, built into many programming languages, is an automatic way of managing the memory available to various objects. The garbage collection mechanism tracks objects still being used, marks the objects no longer in use as garbage and through the process of garbage collection it automatically frees up the memory for use by other objects. In the case of Java, garbage collection is done within the Java Virtual Machine environment.

Garbage collection overhead refers to the additional resources, processing time, used by the garbage collection mechanism.

Software applications can choose to implement various **garbage collection types** depending on the performance goals. In Java, for example, developers can choose to implement garbage collection of type single threaded (serial), multi-threaded (parallel), concurrent mark then sweep (CMS) or parallel collection in multiple memory zones. Each method uses various resources and CPU time, affecting the overall application performance.

The **heap**, for example the **Java heap**, is a repository of Java objects. The heap is the sum of active objects, dead objects (marked for garbage collection) and free memory. **Heap size tuning** in Java refers to minimizing the time that the Java Virtual Machine (JVM) spends doing garbage collection while maximizing the number of clients that the application server can handle at a given time. A **heap map** provides the memory details of a given process.

WebLogic Server is Oracle's Java EE application server, part of Oracle Fusion Middleware suite of products, used for building and deploying enterprise applications.

A **WebLogic Server cluster** consists of multiple WebLogic Server instances running simultaneously and working together to provide increased scalability and reliability.

A WebLogic **domain** is a logically related group of WebLogic Server resources. Domains include a special WebLogic Server instance called the **Administration Server**, which is the central point from which you configure and manage all resources in the domain. Usually, you configure a domain to include additional WebLogic Server instances called **Managed Servers**. You deploy Web applications, EJBs, and other resources onto the Managed Servers and use the Administration Server for configuration and management purposes only.

Oracle Home refers to a directory where Oracle products are installed, pointed to by an environment variable. Multiple active Oracle Homes can exist on the same host.

Oracle **Java Virtual Machine (JVM)** is a standard, Java-compatible environment that runs any pure Java application. It supports the standard Java binary format and the standard Java APIs.

Servlets are Java programming language classes that dynamically process requests and construct responses.

2

Isolate and Diagnose Application Performance Issues

Using Oracle Application Performance Monitoring, you can monitor performance of your application by following transactions across servers to identify the exact tier causing an application issue, see if the issue is specific to a geography and see application logs automatically in context of the application performance. Synthetic Monitoring helps in simulating a path in the application that a user would normally take, and ensure that the user can transition through the different web pages in the path smoothly. This helps is recognizing application performance issues before the end user experiences it.

Topics

Typical Workflow for Isolating Application Performance Issues

This section uses an example scenario to illustrate how you can isolate application performance issues. In this example scenario, as a DevOps administrator, you're responsible for administering and supporting one of your enterprise applications used by your customers interested in carpool and vanpool services. Your line of business executives see a sudden drop in sales on your company website, and they ask you to investigate the reasons for drop in sales on the website. The ordering application is critical to your business, and it's used by your customers daily to place service orders on your website.

Enterprise application deployments are complex. They involve various software tiers comprising applications, databases, web servers, and so on. You need simple and effective ways to isolate application issues and troubleshoot problems quickly.

You start troubleshooting this specific problem by:

- Viewing alerts to see if the average response time for any page has exceeded the threshold
- Checking if the errors are specific to any geography and drilling down to isolate the exact problem location
- Isolating the problem down to the application servers and databases
- Drilling down to logs to determine the exact root cause of the problem causing drop in sales on your website

Here are the common tasks to isolate application performance issues.

Task	Description	More Information
View alerts	From the list of alerts, pick an alert and view details.	View Alerts
Troubleshoot a slow page	View details about the page to identify a possible problem in the page.	Troubleshoot a Slow Page
Identify a slow request	Identify which request is slowing down the performance of the application.	Drill Down to Server Request Details
Identify issues in associated tiers	Inspect associated tiers and identify issues.	Find Issues in Associated Tiers
Identify issues based on location	Identify if issues are being seen in a specific country	Using Geomaps to Find Issues in Pages
View related logs	Drill down to related logs to identify issues.	Drill Down to Related Logs
View Diagrams to spot issues	Study Diagrams to easily spot issues	Isolate Issues through Diagrams

View Alerts

Oracle Application Performance Monitoring notifies you of application performance issues. As a DevOps administrator, you start troubleshooting by viewing such alerts, which provide a starting point to isolate the problem.

You can view alerts from the Alerts page and also from the entity page on which the alert was created. Alerts are created for fixed thresholds or anomalies for metrics on Pages, AJAX calls, and Server Requests.

View alerts from the Alerts page

1. In the Oracle Application Performance Monitoring home page, in the left navigation pane, click **Alerts**.

   OR

   In the Alerts tile on the Home Page, click the number of Alerts. You can also click the number of Critical Alerts, or Warnings to view only the specific alerts.

   The Alerts page displays all the alerts that need your attention.

2. Select *APM* in the **Service** dropdown to view Alerts from Oracle Application Performance Monitoring. You can further filter based on severity. You can view the following details of an alert.

Detail	Description
Message	The last alert message seen on this object. Click the message to view details and history of the alert.
Entity and Entity Type	The object for which the alert exists. Click the entity to open the object. Entity type is the type of object, a **Page**, or an **AJAX call**.
Duration	Duration for which this alert has been open.

3. The icon in the first column indicates the type of alert — the status of an alert could be Critical, Warning, Clear (closed), and Fatal being the most severe. Click the arrow next to the icon to view some more details of the alert like Created Date, the Updated Date, and a brief history. In case of a closed alert, the Closed Date is displayed.

4. Click the alert Message to view the details of the alert more closely.
5. Click the Entity to view details of the entity on which the alert was created.

View alerts from the entity page

If an alert is created on an entity based on alert rules, the alert will be displayed in the entity page for that time period. You can view the alerts in the Alerts pane on the entity page.

- The Alert pane displays the number of new alerts, open alerts that were carried over from previous time periods, and alerts that are still open at the end of the selected time period.
- The Alerts tab lists all the alerts during the selected time period and their current status. Click the arrow next to the status of the alert to view details.
- If the alert is an early warning, a chart indicates when the trigger event occurred. A prediction of how soon an error might occur is indicated.

Troubleshoot a Slow Page

Oracle Application Performance Monitoring helps identify a page that is loading slowly and points to possible reasons for the decrease in speed.

With Oracle Application Performance Monitoring, you can get an insight into how your application is performing at the user-end. The performance of your application's web pages are monitored, and if your users are experiencing issues with any specific page, Oracle Application Performance Monitoring alerts you. With Oracle Application Performance Monitoring, you can also diagnose the reason for a slow page, and identify if the actual issue is at the server level or with the browser.
To identify the reason for a slow page:

1. You can start from the **Pages** tile in the home page or Pages list.

   The Pages list gives some quick information about the listed pages. Below the page icon, you can see which browser was used. The Apdex value indicates the overall performance of the application. The Page User Satisfaction graph indicates the user experience with the page. The Page Load Time and the Views and Errors provide further data points on page performance.

2. If a particular page, say **cart.jsp** seems to be taking an unusually long time to load, as indicated by *Average Load Time* or *Max Load Time*. Click the page name to view page details.

   The *Alerts* pane shows the number of alerts for the page for the selected time period. The *Apdex* pane shows the Application Performance Index for the selected time period, along with a chart for user satisfaction. The *Average Page Load Breakdown* chart indicates which phase of the page load is taking a long time. In this example, you can see that loading the *First Byte* is taking the most time, indicating a server side issue. This needs to be inspected further.

If you see a break in the Page Load Time graph, it could be because the Page was not used during that time period, and hence no data was recorded. This is applicable to all entities.

3. You can diagnose performance of AJAX calls seen on the page. Click the **Ajax Calls** tab to view the list of AJAX calls on the page, and view details for any AJAX call with slow response time.

   AJAX requests are automatically correlated with their page details and server requests for rapid identification of problems.

4. Click the **Server Requests** tab to see which server requests were called to serve the page. If a server request appears to be slow, you can click the server request to view details and locate an issue, if any.

5. Click the **Instances** tab to view instances of pages and to navigate to sessions related to those page instances. You can diagnose performance issue or error in the context of a user session from here.

6. Click the **Alerts** tab to view all the alerts which were active for the page during the selected time period. Click an alert to view history of the alert.

Drill Down to Server Request Details

Oracle Application Performance Monitoring automatically discovers, classifies, and measures all your server requests. You get the information you need to understand what tier, request, and operation the application issue resides in. Let's start monitoring server requests to investigate which requests have issues.

To identify the reason for a slow request:

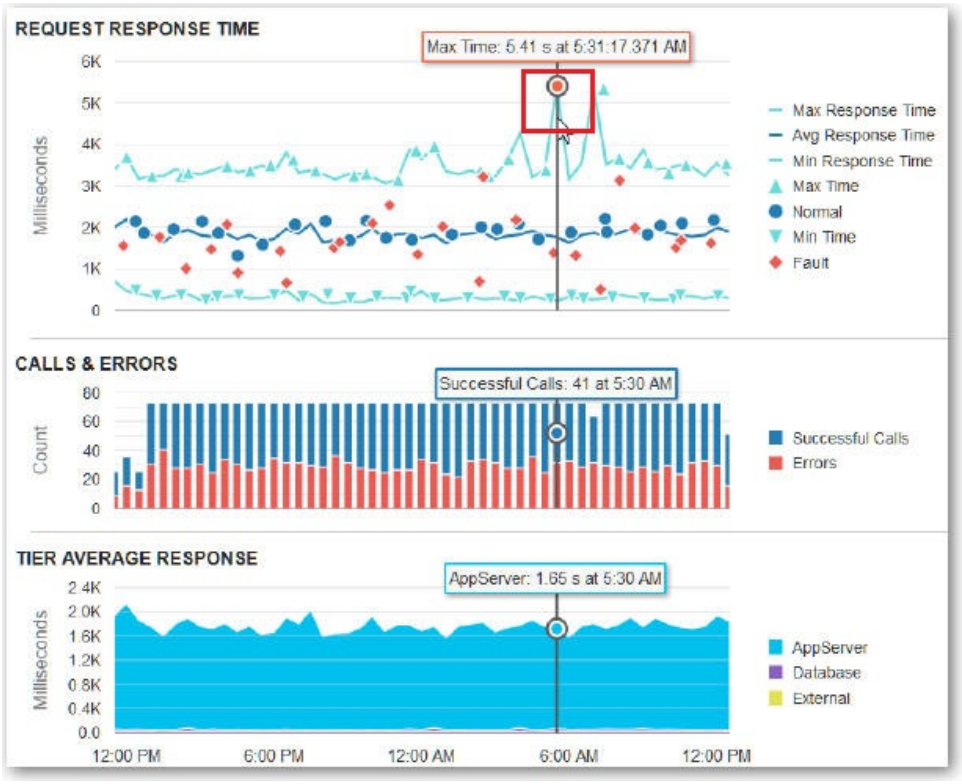
1. Monitor the response time for top five requests in the home page, or across all the requests in the Server Request list.

2. If you find that the response time of a particular request is more than what's acceptable, then view the details of the request.

   In this example, the request **checkout** seems to have a high *max response time* of over 5 seconds, and over 30% errors. This means that on an average, users are not seeing this high *max response time*, but at least some users are experiencing up to over 5 seconds of *max response time*. Let us inspect this further.

3. Click to view details of the server request.
4. In the **Diagram** tab, see a pictorial representation of the calls made by the server request. You can see how many calls the request has received, how many internal and external calls it has made and if there are any errors in any of these calls. Hover over the objects in the diagram to view details in the **Tooltip** pane.
5. Click **Metrics** to view further details of the server request.

   You can see in this example, that the *max response time* of the request has peaked to 5.41 seconds at 5.31 AM. This is higher than the expected value of 2 seconds.



6. Drill down to the details of the application server to check for any issues resulting in a slow request.

7. Inspect further to see if an associated tier is causing a slow request. Drill down to the **Database** tab and see if the issue is originating in the Database tier.

8. Click the **Alerts** tab to view all the alerts which were active for the server request during the selected time period. Click an alert to view history of the alert.

The details displayed in the Server Request page, the contextual details displayed in the Application Server page, and the logs for the application server will help you identify the reason for a slow request.

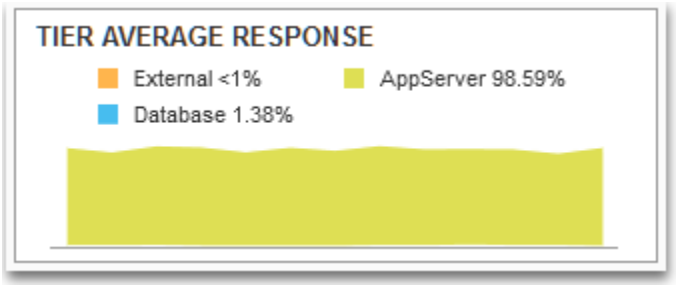
Find Issues in Associated Tiers

Oracle Application Performance Monitoring helps you recognize bottlenecks in tiers associated with server requests.

To isolate the application performance problem down to the application infrastructure, such as database or application servers, navigate from the server request to the **Database** tab or to the Application Server page. To recognize issues in associated tiers:

1. Identify a slow request and view the request summary.
2. Place the cursor at any time point on the **Tier Average Response** graph to view the response time of each tier at that point. The **Tier Average Response** cell displays the various response times of each tier — the **App Server**, the **Database**, or the **External Tier**.

   In this example, the bulk of the average response time is spent on processing in the application server tier. The **Tier Average Response** graph also shows how the tiers trend over time - in this example you can see that in the selected time range, the application server tier is consistently contributing more to the average response time than the other tiers.



   In this example the request's response time has peaked to 5.41 seconds at 5.31 AM. This is higher than the expected value of 2 seconds.

3. In the **Diagram** tab, see if all operations in the application server are slow, or if one particular operation is slow.
4. The **Application Sever** tile shows a view of the performance of application server. Drill down to the application server page to check if the issue is in the application server tier.
5. Drill down to check if the issue is in the database tier. To diagnose issues with the database tier, go to the **Database** tab to analyze the specific SQL statements and view the database logs.
6. If both the tiers do not seem to have an issue, check the external tier for possible issues.
7. Click the **Instances** tab to view details of a specific instance. You can drill down into the logs of an instance to get more details on the faults.

   Here's an example of the faults in an instance.

Response Time (ms)	Self Time (ms)	Latency (ms)	Faulty	Call Type
1,396	6	0	⊗	⟷
7	3	2		⟷
1,383	77	6	⊗	⟷

You can also see these faults in the **Diagram** tab, and on the **Links** tab.

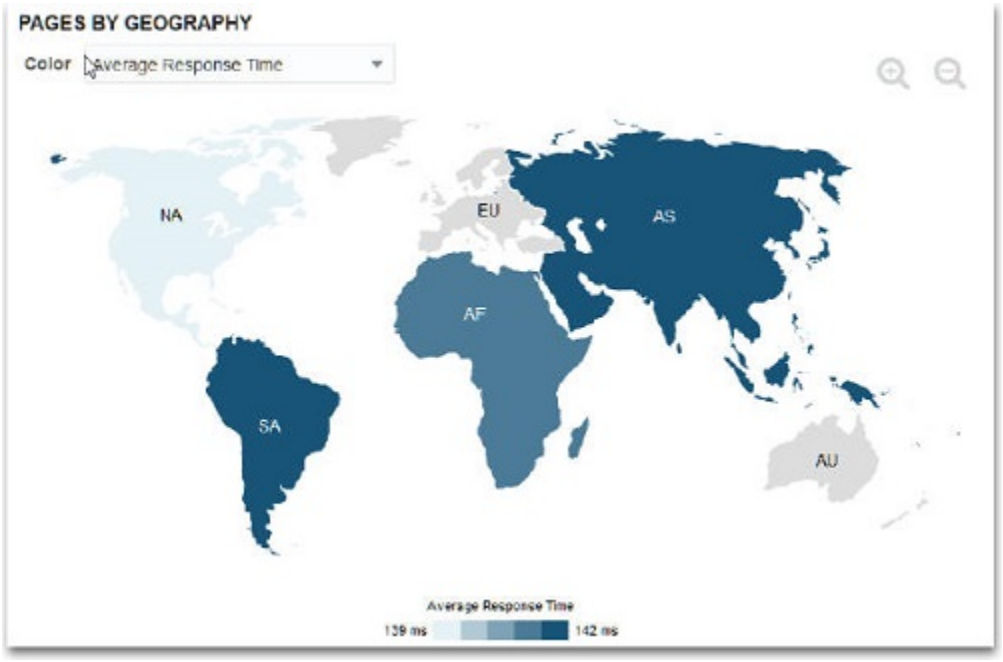
Find Issues in Pages Using Geomaps

Oracle Application Performance Monitoring helps you isolate issues in pages based on geography.

You can inspect if any issues in pages are cropping up in a certain location through the **Pages By Geography** pane in the home page. To isolate issues related to pages in a specific geographic location:

1. In the Oracle Application Performance Monitoring home page, select a measurement by which you want to see pages, in the **Pages By Geography** pane.

   The color coded map indicates the locations where page usage has been observed and measured.



2. Click a color coded continent and click the *Zoom In* icon to drill down to the countries where the selected measurement is further applied.
3. Select a country within the continent and click the *Zoom In* icon to view details specific to the country.

The countries are color-coded based on the values of the metric selected in the drop down, with the lightest being the least number of pages of a selected metric. Hover over a country to view all measurements Oracle Application Performance Monitoring has observed for page-views within that country.

List of countries in the Geomap

You can drill-down and view regions within the following countries in the geomap:

- 'BEL': Belguim
- 'CHN': China
- 'FRA': France
- 'DEU': Germany

- 'GBR': Great Brittain / United Kingdom
- 'IND': India
- 'ITA' : Italy
- 'JPN': Japan
- 'ESP': Spain
- 'THA': Thailand
- 'NLD': The Netherlands / Holland
- 'USA' : United States of America

Drill Down to Related Logs

To isolate the application performance problem further, you can view and inspect log events of a request instance or an application server that might be causing the problem.

You can view logs from:

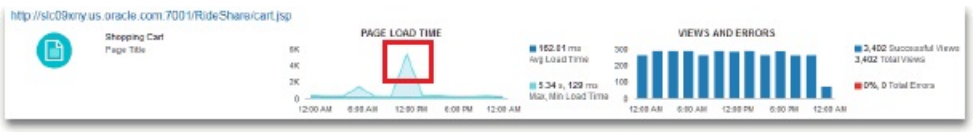
- Application Server details page to see logs for that one application server.
- Server request details page to see logs for the application server and databases relevant to the server request.
- Server request Database tab to see logs for the databases relevant to the server request.
- Server request Instances to see logs for the application server and database(s) specific to that server request instance.

To view log events of a server request or an application server:

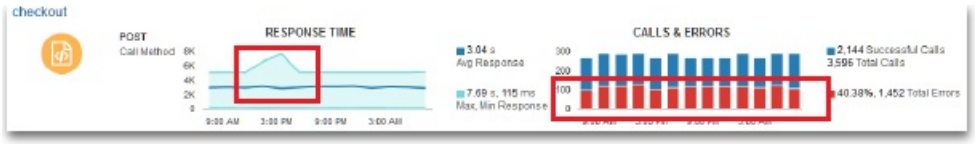
1. Drill down to logs related to a server request:
    a. In the Server Request Details page, go to the **Instances** tab and select an instance.
    b. In the Server Request Instance page, click **View Related Logs** above the summary pane.
2. Drill down to logs related to an application server:
    a. Select the application server and view details.
    b. Click **View Log** above the Application Server summary pane.

Here's an example of how to drill down to related logs to isolate an issue.

1. Let us start from the `cart.jsp` page, for which an alert was displayed, primarily because the response time for the page was very high.

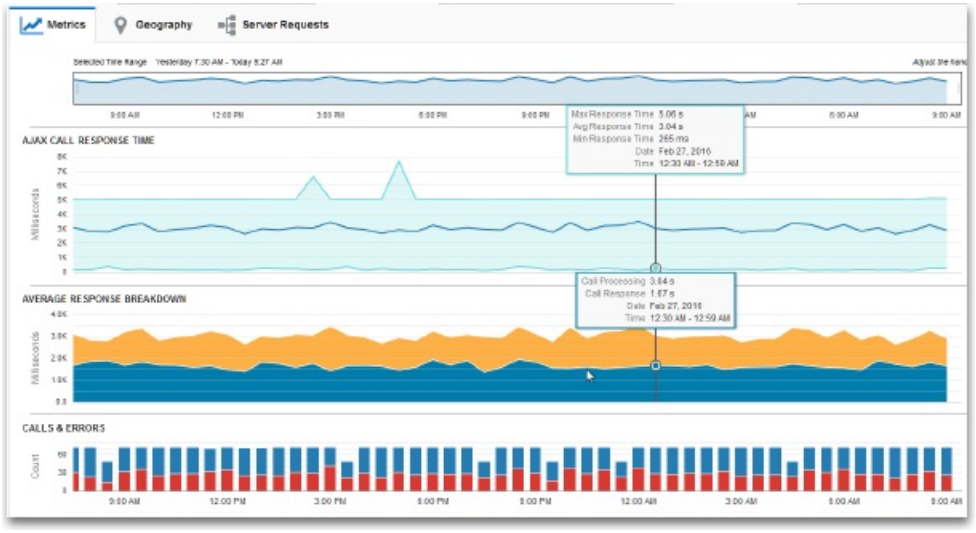


2. Drill down to view details of the **checkout** Ajax call.

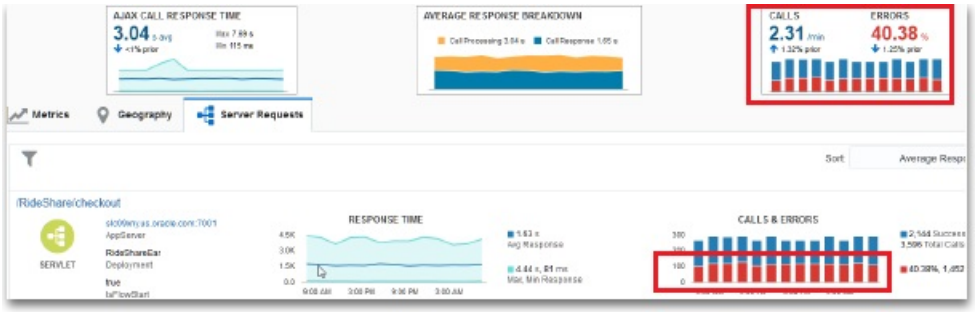


Notice that the Ajax call has encountered a high number of errors.

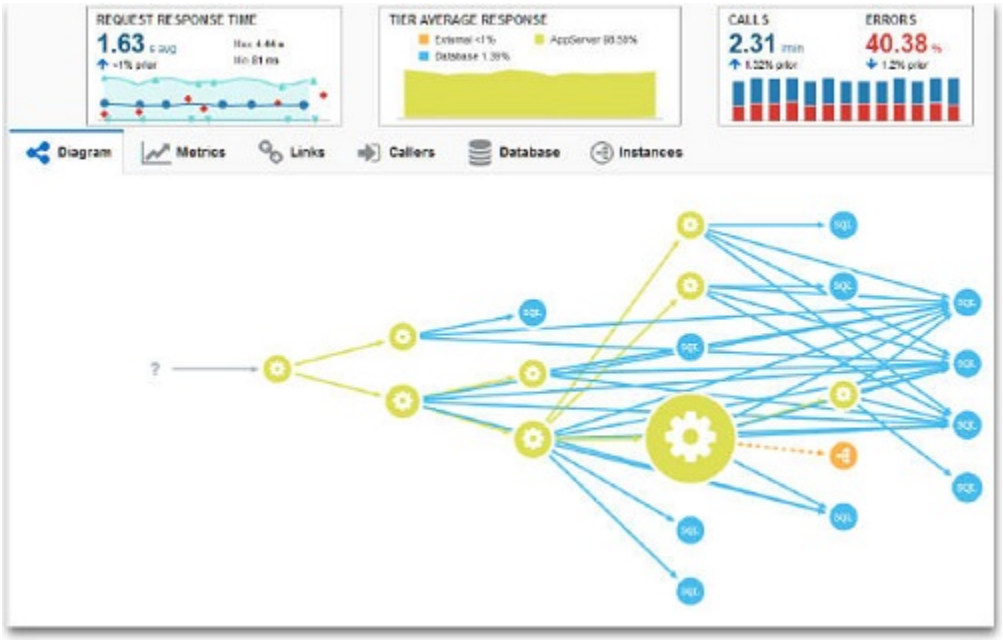
3. The call processing and the response times indicate a very slow call along the timeline. The call has encountered some errors.



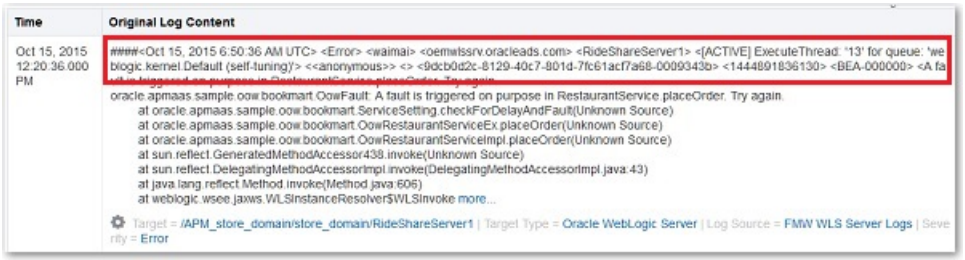
4. The corresponding server request **checkout** indicates errors. Let us drill down further to view details of the server request **checkout.** The errors seem to be very high, close to 40%.



5. The **Diagram** tab displays all calls made by the server request. Hover over an object or an arrow to view details of the object or the call.



6. Let us drill down to Instances to see what operation is failing. In the **Instances** tab, pick an instance which has a fault. Click **View Related Logs** to inspect this further by viewing logs.
7. The log points to the time when the fault occurred, and indicates issues at the application server and the database levels.



Isolate Issues through Diagrams

The Diagram tab in the Server Request details page gives a quick diagrammatic view of all the objects associates with the server request.

Here is the diagram of a server request with all the connected objects like the SQL calls, server requests and AJAX calls. The question mark indicates an unknown caller.

Using the Diagram

The diagram represents the server request in the center, with all the calls made to and from the server request represented by a node. Hover over any connector between two nodes to cut out other traffic, and view details about the specific call. Hover over any node to view only the connections to and from the selected node. This helps in isolating the specific call you are looking for, to enable quicker identification of issues. Here is example of how hovering over a connector and a node cuts out other information from the diagram.

Using the Calls table

The Calls table that appears below the diagram lists all the calls made to and from the server request, showing only information pertaining to the object currently selected in the diagram. You can drill down further from this table to view the details of the server request, or of a related AJAX call to isolate a problem.

Using the Context menu

You can right-click on any node in the diagram to see a context menu through which you can easily move forward with troubleshooting and isolating the cause for an issue. The options available in the context menu depends on the type of object the selected node represents.

For example, right click a SQL call and select **Isolate this Operation's Calls**. This will remove all other nodes from your diagram. From among the existing nodes, click on a server request node to display the operation's inward and outward paths.

Typical Workflow for Using Synthetic Monitoring

You can use Synthetic Monitoring to script or record user paths, and use this to simulate user transactions on the application. These paths can be continuously monitored through Application Performance Monitoring, and potential issues can be caught early, before the end user experiences it.

> **Note:**
>
> You can define and use Synthetic Monitoring only if you have installed the Cloud Agent on Linux.

Here's a typical workflow for setting up and using Synthetic Monitoring:

Task	Description	More Information
Deploy Cloud Agents	This is a requirement before you can define locations. This is applicable only for private locations.	See Install Cloud Agents in *Installing and Managing Oracle Management Cloud Agents.* To ensure that you can define and use Synthetic Monitoring, the Cloud Agent should be installed on Linux.
Check for pre-requisites	Review the list of pre-requisites. This is applicable only for private locations.	Pre-requisites for Locations
Define Locations	Define locations. This is done by an APM Administrator. This is applicable only for private locations.	Define Locations
Define Synthetic Tests	You can define synthetic tests for a HTTP Ping, Page Load or a Scripted Action.	Define Synthetic Tests
Review Synthetic Test reports	Use the Synthetic Test reports to monitor the performance of your applications.	Monitor Application Performance through Synthetic Tests
View Sessions	For synthetic tests of type Scripted Actions, you can view details of the session when the test was run. This option is available only if you are running synthetic tests on an application that is also being monitored by Oracle Application Performance Monitoring.	• Navigate to Sessions from the Instances tab. See Monitor Application Performance through Synthetic Tests • Monitor End User Experience through Sessions
View HAR Reports	View HAR reports for HTTP Ping or Scripted Action. This is available for public locations.	• View HAR reports from the Instances tab. • Download HAR files. See Monitor End User Experience through Sessions.

Define Synthetic Tests

You can schedule synthetic tests for various locations and ensure that the performance of the application is monitored at all times.

To define a synthetic test:

1. In the left navigation pane, click **Administration** and select **Synthetic Tests**.
2. In the **Create Synthetic Monitoring Test** window, choose the **Type** of test to create.
   - HTTP Ping — Testing the connectivity to and performance of your application
   - Page Load — Testing the performance of a single URL, being loaded by a browser
   - Scripted Actions — Testing the performance of a complete workflow recorded using Selenium scripting.
   - Rest Web Service — Testing the performance of a complete workflow that uses REST web service.
3. If you are creating a synthetic test of the type **HTTP Ping**, provide these additional details:
   - Name: Provide a name for the synthetic test you are creating.
   - URL: Select HTTP or HTTPS and specify the URL you want to test.
   - Location: Choose the location/s from where you want to run the test.
   - Application: Optionally, select an application within which the result of this test will be displayed. Associating the test to an application ensures that the test results and alerts will be visible with the application reporting context in the Oracle Management Cloud UI.
   - Frequency: Specify at what interval you want the test to be executed.
   - Verify certificate: Check this option if you want to verify the validity of the SSL certificate during the tests.
   - Redirect: Check this option if you want the test to fail in case there is a redirection.
4. If you are creating a synthetic test of the type **Page Load**, provide these additional details:
   - Name: Provide a name for the synthetic test you are creating.
   - URL: Select HTTP or HTTPS and specify the URL you want to test.
   - Location: Choose the location/s from where you want to run the test. Private locations defined by your administrator, and public locations that are configured are listed here.
   - Application: Optionally, select an application within which the result of this test will be displayed. Associating the test to an application ensures that the test results and alerts will be visible with the application reporting context in the Oracle Management Cloud UI.
   - Frequency: Specify at what interval you want the test to be executed.

5. If you are creating a synthetic test of the type **Scripted Actions**, provide these additional details:
   - Name: Provide a name for the synthetic test you are creating.
   - Base URL: Select HTTP or HTTPS and specify the base URL on which to run the test. In the script, the Base URL will replace the URL from where you have recorded the Selenium test.
   - Select File: Click **Choose File** to browse and select the Selenium script for the synthetic test. The selected file can be of the format `.java` or `.side`.

     > **Note:**
     >
     > a. If it is a `.java` file, ensure that the script is exported from Selenium either as a Java JUnit for WebDriver or a Java TestNG file. Ensure you run the complete recording in Selenium before exporting the script.
     >
     > b. If you are creating a `.side` file, note that you can create Test Suites using Selenium IDE. Ensure that your `.side` file contains only one test, and that you have run the complete recording in Selenium before exporting the script. To see the list of supported Selenium commands, see Supported Selenium Commands in Synthetic Tests.

   - Optionally, click **Preview File** to view the contents of the uploaded script. Note that any edits to the script should be done through Selenium IDE or other preferred tools.
   - Location: Choose the location/s from where you want to run the test. Private locations defined by your administrator, and public locations that are configured are listed here.
   - Application: Optionally, select an application within which the result of this test will be displayed. Associating the test to an application ensures that the test results and alerts will be visible with the application reporting context in the Oracle Management Cloud UI.
   - Frequency: Specify at what interval you want the test to be executed.
6. If you are creating a synthetic test of the type **Rest Web Service**, provide these additional details:
   - Name: Provide a name for the synthetic test you are creating.
   - URL: Select HTTP or HTTPS and specify the REST URL on which to run the test.
   - Location: Choose the location/s from where you want to run the test. Private locations defined by your administrator, and public locations that are configured are listed here.
   - Application: Optionally, select an application within which the result of this test will be displayed. Associating the test to an application ensures that the test results and alerts will be visible with the application reporting context in the Oracle Management Cloud UI.
   - Set the time Interval and Request Time Out.

- Select Authentication if required.
- For **Request Configuration**, select a **Method** — either GET or POST. Specify the Query and Header parameters as required.
- For **Response Configuration**, specify the Expected Http Status Code. Check the Verify Content option, and specify a regular expression to validate the response output as required.
- Select the **Redirect is Failure** option if the test should fail on redirection.

7. Click **Save**.

The schedule will be displayed in the list of Synthetic Tests, and will run as per the frequency specified in the schedule. You can edit a saved synthetic test by clicking the name of the test.

Monitor Application Performance through Synthetic Tests

You can monitor the performance of your application through synthetic tests and identify possible issues before they occur.

Oracle Application Performance Monitoring enables you to define and run synthetic workflows on your application. You can define a test for a HTTP ping, test for a specific page, or record a Selenium based script of a workflow on your application, and run these monitoring tests on your application anytime without having to wait for the actual workflow to occur.

You can view the results of the scheduled test and monitor the performance of the application, view the usage of resources and isolate possible issues.

To view the reports of scheduled synthetic test:

1. In the left navigation pane, select **Synthetic Tests**. All the scheduled synthetic tests are listed in the Synthetic Tests pane.

   From this pane, you can view high level data about the listed synthetic tests, like the type of test, application, location, frequency, execution time, and availability. Scan through these details to identify the synthetic test you would like to drill down into.

2. You can sort the listed synthetic tests on a number of criteria. Sort the tests based on **Status** to view the tests with errors on top. A green check mark over the test icon indicates a successful test, and a red X indicates errors while running the test.
3. Examine the metrics and select the synthetic test report to drill down into. Click the synthetic test. The Synthetic Test page displays details of the synthetic test.
4. The **Metrics** tab displays information on availability, execution time, time breakdown, transfer rate and download size. The details in this tab are for all the tests executed across all locations, and depends on the type of synthetic test.
   - HTTP Ping: This report displays information like availability, execution time, transfer rate, download size and ping time breakdown.
   - Page Load: This report displays information like availability, execution time and total load time breakdown.
   - Scripted Action: This report displays information for multiple pages that are part of the script and includes data points like AJAX calls and total load time breakdown.

5. The **Instances** tab displays details for individual tests that were run across all locations.

   You can view the status of the test run at a specific time, for a specific location. If there is an error in the test, the error message is displayed in this pane.

   - For synthetic tests of type Scripted Actions, you can view details of the session when the test was run. In the **Instances** tab, click **View Session**. This option is available only if you are running synthetic tests on an application that is also being monitored by Oracle Application Performance Monitoring. The Session page displays a timeline view of the session along with details of multiple pages accessed during a user session. You can further drill down into the details of the individual pages within the timeline. See Monitoring End User Experience through Sessions.
   - For synthetic tests created on public locations, and for of the type Scripted Action and Page Load, you can view HAR reports. In the **Instances** tab, click **View Har**.

     The Har Statistics page displays details of the HTTP pages. You can view a summary of the data as graphs and detailed tables.

     > **Note:**
     >
     > On Firefox, if you are trying to view HAR files, the browser might display an error '**Unresponsive Script**'. Click **Continue** and wait for the script to complete. This usually happens when the HAR files are large (above 600 KB).

   - To download the content, click **Download Har** or **Download Screenshot**. When downloading the content as a screenshot, it will be downloaded to the local host as a zip file.

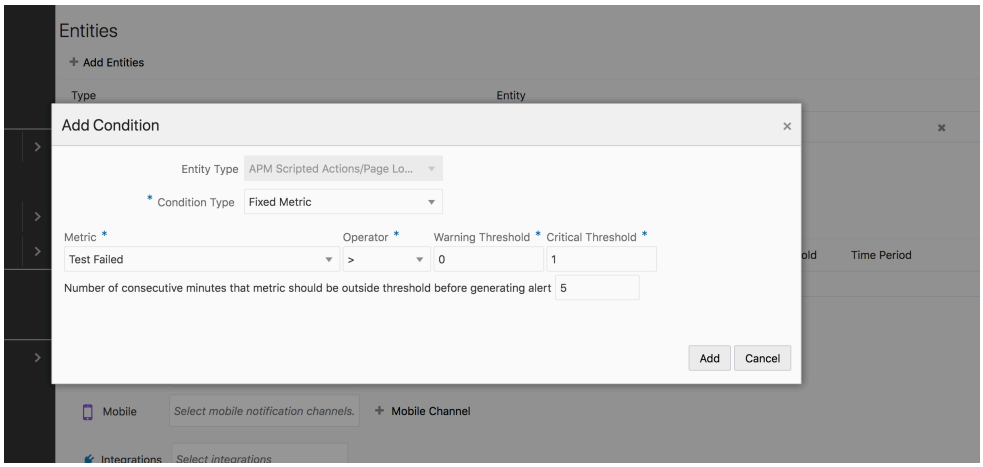
Create Alert Rules Based on Synthetic Tests

Create Alert Rules when selected Synthetic Tests meet defined conditions and send a notification when the alert is raised, worsens in severity, or is cleared.

To create a synthetic test alert rule:

1. From the left-hand navigation, click **APM**, then select **Alert Rules**.
2. Click **Create Alert Rule**. Enter a name and click **Add Entities**.
3. In the **Select Entities** menu, select **Individually** and click a Synthetic Test.
4. Click **Add Condition** and select a **Test Failed** metric with a warning or critical threshold greater than 0. Click **Add**.
   The "number of consecutive minutes that metric should be outside threshold before generating alert" dialog should be less than the Collection Frequency selected for the Synthetic Test.

Figure 2-1 Test Failed Synthetic Alert Rule



5. Create a new condition with the same parameters for the **Test Error** metric.
6. Add the required notification channels.
7. Click **Save**.

Metric and Frequency Examples:

- **Number of consecutive minutes >= Test frequency:** This will generate an alert on the first test failure.
- **Number of consecutive minutes >= 2*Test frequency:** This will generate an alert on two consecutive failures.
- **Test Failed > 0:** This will evaluate a warning alert when there is a test failure.
- **Test Failed >= 1:** will evaluate true when there is a test failure.
- **Test Failed > 1:** will Not evaluate true when there is a test failure as on Failure metric value will be "1"

  Test Failed > 0.5: will evaluate true when there is a test failure.

To learn more about Alert Rules, see Create Alert Rules.

Troubleshoot Synthetic Tests

If you run into problems while using Synthetic Tests, here are some tips to debug.

Debug Cloud Agent Location

You can run Synthetic Tests on a private or a public location. For a test to run successfully on a cloud agent, a few basic set of prerequisites, called *Location Compatibility* have to be in place. To check for Location Compatibility:

1. On the Oracle Management Cloud home page, click **APM**. In the left navigation pane, select **APM Admin**, and then, **Locations**.
2. For the required location, click **Compatibility Check**.

   A green tick mark indicates that all the prerequisites are met; else a warning or error is displayed.
3. Click on the status indication icon to view these details.

a. **Agent Version** — Indicates the version of the cloud agent. Ensure that the Cloud Agent version is 1.33 or higher.

b. **Firefox Version** — Indicates the version of the browser. Ensure that you have the correct Firefox version for your system to successfully execute the Selenium tests.

   - Oracle Linux 6: Firefox version 45
   - Oracle Linux 7: Firefox version 61-66

   > **Note:**
   >
   > Firefox is the only supported browser. Other Firefox versions including Beta versions are unsupported.

   You can check if Firefox is present on the cloud agent machine by running the command `firefox --version`.

c. **Proxy Status** — Indicates the status of the proxy. Ensure that the proxy specified is correct and reachable. Edit the location to correct the proxy information, if required.

d. **Proxy Error Message** — Displays an error message in case of an error in the proxy settings.

e. **X-Server Unavailable Ports** — Indicates the X-Server ports that are not available. Create X-Server on the ports that are missing.

f. **X-Server Unavailable Ports Message** — Displays an error message if there are unavailable X-Server ports.

Debug Cloud Agent Crash Due to Memory Issues

When running Synthetic Tests that generate big HAR files, the Cloud Agent may run into memory issues and crash. When the Cloud Agent crashes, a log file is generated with name: `hs_err_pid<pid>.log`. It should look like the following:

```
Stack: [0x00007f771c697000,0x00007f771c798000],
sp=0x00007f771c795220,  free space=1016k
    Native frames: (J=compiled Java code, j=interpreted, Vv=VM code,
C=native code)
      C  [libzip.so+0x11d10]  newEntry.isra.4+0x60 C
[libzip.so+0x12b57]  ZIP_GetNextEntry+0x37
      J 3024  java.util.zip.ZipFile.getNextEntry(JI)J (0 bytes) @
0x00007f776995def6 [0x00007f776995de40+0xb6]
      J 1477 C1 java.util.zip.ZipFile$ZipEntryIterator.next()Ljava/
util/zip/ZipEntry; (212 bytes) @ 0x00007f77694d6b4c
[0x00007f77694d68a0+0x2ac]
      J 1475 C1
java.util.zip.ZipFile$ZipEntryIterator.nextElement()Ljava/lang/Object;
(5 bytes) @ 0x00007f77694d5f84 [0x00007f77694d5ec0+0xc4]
    j
oracle.sysman.emd.fetchlets.gfmsynmon.common.CommonUtil.addToZipfile(Lja
va/io/File;Ljava/lang/String;Ljava/lang/String;Ljava/lang/
String;)Ljava/io/File;+106
    j
```

```
oracle.sysman.emd.fetchlets.gfmsynmon.selenium.HarFetchletUtils.prepareC
ombinedZip(Ljava/io/File;)Ljava/io/File;+349
    j
oracle.sysman.emd.fetchlets.gfmsynmon.selenium.HarFetchlet.getMetric(Lja
va/util/Properties;Ljava/util/ArrayList;Loracle/sysman/emSDK/agent/
datacollection/CollectionFactory;Loracle/sysman/emSDK/agent/fetchlet/
FetchletContext;Loracle/sysman/emSDK/agent/TargetID;Ljava/util/
Map;Loracle/sysman/emSDK/agent/fetchlet/StateFullCallbacks;)Loracle/
sysman/emSDK/agent/datacollection/CollectionResult;+428
    j
oracle.sysman.gcagent.target.interaction.execution.FetchletFactory.getMe
tric(Ljava/util/Prop
```

To solve this issue, follow these steps:

1. Navigate to your Cloud Agent installation folder, and edit **emd.properties** with a text editor.
2. Search for the `agentJavaDefines` property and add the following flags:

   ```
   agentJavaDefines=-Xmx2G -XX:MaxPermSize=128M -
   Dsun.zip.disableMemoryMapping=true
   ```

> **Note:**
>
> The `-Xmx2G` flag assigns 2GB as the maximum memory allocation pool for a Java Virtual Machine (JVM). The `-Dsun.zip.disableMemoryMapping=true` flag is needed for Cloud Agents versions 1.49 and below.

Debug Test Execution

You can check for the status of synthetic tests, and debug if they are not getting executed properly by following these steps.

1. Create a synthetic test with a private or a public location. Wait for a few minutes before checking for its deployment. A test on a private location takes about 5 minutes to deploy, and about 15 minutes to deploy on a public location.
2. On the Oracle Management Cloud home page, click **APM**. In the left navigation pane, select **APM Admin**, and then, **Synthetic Test Definitions**.
3. For the required location, click **Check Deployment**. The status of the test is displayed in the Test Status dialog box.
   a. **Location Name** — Indicates the name of the private or public location.
   b. **Deployment Status** — Indicates whether the test got deployed on the Agent or the Cloud Container.
   c. **Last Run Status** — Indicates the time the test was last run. If the test was not executed, check for its location compatibility.
   d. **Last Deployment Time** — Indicates the last time the test was deployed onto the Agent or the Cloud Container.

This time is first recorded when the test is created, and updated for each edit of the test. If the **Deployment Status** is *Failed*, and the test **Run Status** is *Successful*, then it means that the last update of the test failed.

Debug Log Location

You can check the logs to diagnose the failed test execution by following these steps:

1. Change directory to the `agent_inst` folder:
   ```
   $ cd $AGENT_HOME/agent_inst
   ```
2. Check test name in `emd/targets.xml` and note `test_meid`:
   ```
   <Property NAME="test_meid" VALUE="06E1665FE0A82B8057506B2A45F8FFC6"/>
   ```
3. Check logs in `test_meid/log folder`:
   ```
   $ cd $AGENT_HOME/sysman/ApplicationsState/beacon/
   06E1665FE0A82B8057506B2A45F8FFC6>/log/*
   ```

3

Monitor Application Performance

End users of your applications usually complain about the speed of retrieving data from web pages and general page response times. You need ways to monitor several aspects of web pages and their interactions. This data helps you find and resolve issues before your end users complain.

Topics

Typical Workflow for Monitoring Application Performance

Monitoring Application Performance comprises of monitoring end user experience, performance of Server Requests, Application Servers, and other entities. Here are the common tasks for monitoring application performance.

Task	Description	More Information
Monitor end user experience through performance of pages	Watch the performance of your application's pages and monitor end-user experience	Monitor Page Performance
Monitor end user experience through Ajax requests	See if an Ajax call is the cause for the slow page and is affecting end-user experience	Monitor Ajax Requests
Monitor user sessions	Check if a particular user session is running into errors	Monitor End User Experience through Sessions
Monitor server request performance	See if a server request or an application server is encountering errors	Monitor Server Request Performance
Monitor end user experience based on a web application	Group a set of objects matching defined filter criteria to monitor end user experience	Monitoring End User Experience of a Web Application

Monitor End User Experience

Monitoring the experience of end users of today's applications involves closely watching various aspects of an application, including monitoring of page performance, monitoring of AJAX requests, and monitoring of application request performance.

Let's consider an example of a *RideShare* application. As a DevOps administrator, you need to monitor the performance of:

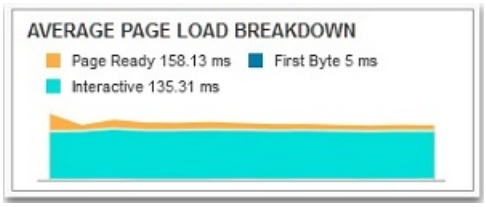
- The page `cart.jsp`, the page where the orders are completed.
- The AJAX call **checkout,** the request that will be called by the page `cart.jsp`.
- The application request **checkout** that is associated with the page `cart.jsp`.

Monitor Page Performance

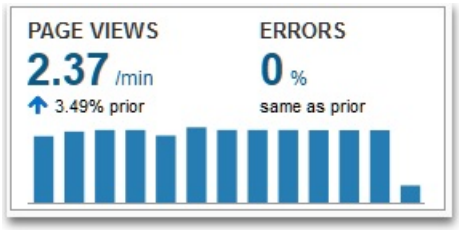
Your customers access web pages in your application, and they expect superior performance. Slow page loads, high page response times, failed transactions frustrate your application users. As the DevOps administrator, you need tools to monitor page performance and get deep visibility into all web pages, transactions, and so on. Oracle Application Performance Monitoring also enables you to track errors that affect page performance.

In the Oracle Application Performance Monitoring home page, the **Pages** tile displays the top five pages. Click the page that you want to view details about.
If the page you are looking for is not among the top five:

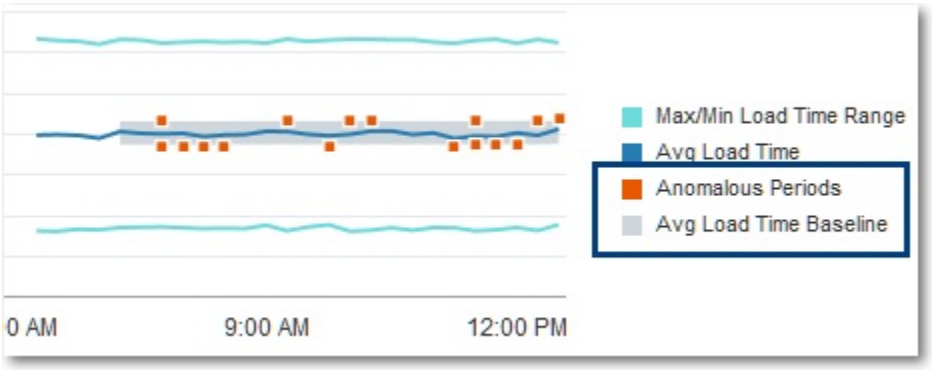
1. In the left pane, click **Pages** to see a list of Pages.
2. Click the name of a page to view the page details.



- In the above example, the page `cart.jsp` started its loading with the first byte at 5 milliseconds, and the page became interactive at 135 milliseconds. The page was completely loaded and ready for customer input at 158 milliseconds, and this is probably an issue because the page load time is high.



- The page was viewed more than 400 times.
- Check the Anomalous Periods and Average Load Time Baseline options to view the baseline and anomalies on the average load time and for page views. The anomalies are depicted as spots above and below the baseline.



To improve grouping of pages, and to enhance the ability to identify performance issues, patterns in URLs that are related to unique numeric identifiers will be replaced by the character '*****'.

Example: The URLs `http://www.oracle.com/01234/page` and `http://www.oracle.com/12345/page` both will be reported as `http://www.oracle.com/*/page`.

Monitor Ajax Calls

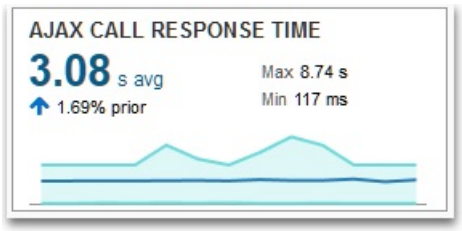
As the DevOps administrator, your goal is to improve and accelerate the performance of your applications. End users don't know what causes slow performance, but you need more visibility into several aspects of your application that affect end-user experience. Oracle Application Performance Monitoring enables you to track Ajax errors in your application. Poor performance of Ajax requests affects the experience of your end users, and it reflects poorly on your website. You should monitor Ajax interactions of web pages in your application, so you can find out and resolve issues.

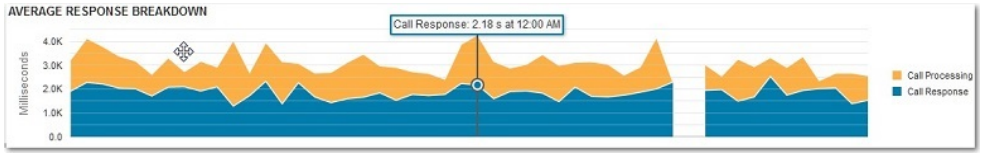
In the Oracle Application Performance Monitoring home page, the **Pages** tile displays the top five pages. Click the page that you want to view details about. If the page you are looking for is not among the top five:

1. In the left pane, click **Pages** to see a list of Pages.
2. Click the name of a page to view the page details.
3. Select the **Ajax Calls** tab and click a call to view its details.

Alternatively, you can also use the **Search** pane to search for a Ajax call directly.

- In the example here, the response time of the AJAX call **checkout** is very slow at over 3 seconds.





- The response time for the call seems to be very high, and this is a concern. In the graphs, you can see that the call response time peaked to 2.18 seconds at 12.00 AM.

- The Successful Calls graph shows the number of successful calls that were encountered through the selected time period.

  Check the Anomalous Periods and Calls Baseline options to view the baseline and anomalies on successful calls. The anomalies are depicted as spots above and below the baseline. You can also see the baseline and anomalies for Ajax Call Response Time.

To improve grouping of Ajax Calls, and to enhance the ability to identify performance issues, patterns in URLs that are related to unique numeric identifiers will be replaced by the character '*****'.

Example: The URLs `http://www.oracle.com/01234/page` and `http://www.oracle.com/12345/page` both will be reported as `http://www.oracle.com/*/page`.

Ajax calls with the same URLs are grouped and can be seen from the Server Requests page. When the same Ajax call is served by different application servers, these will show-up as different server requests. This allows you to compare different application servers handling the same server requests.

Monitor End User Experience through Sessions

Oracle Application Performance Monitoring enables monitoring of end user experience by providing data on user sessions.

The sessions page gives a quick view of the status of every user session. You can sort the sessions based on their health or number of errors, to see sessions with possible issues on top of the list.

You can also see an aggregate view on sessions by location by clicking the **Graph** icon. This functionality also offers slice options by creating your own filters and can help you identify relevant sessions for further diagnosing.

To view a user session:

1. In the left pane, click **Session List** to see a list of user sessions.

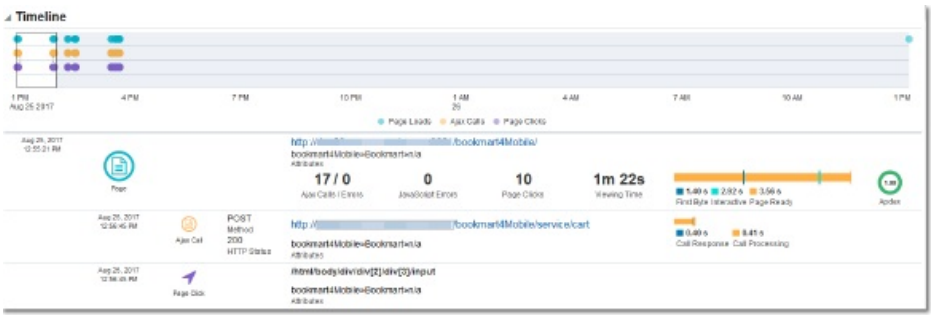
   Scan the list of sessions to see if any session indicates possible issues. Check the indication below each Session icon to see which browser was used. The Session Health indicator and the number of errors will help you decide if any session needs attention.

2. Click the date and time of a session to view its details.

- The Summary at the top of the page gives you a quick view of the session, with a graphical representation of the session health, based on page views rated *Good, Fair* or *Poor*.

- The Timeline displays the various objects spread across the selected period. You can use the slider to view data for the selected time period.

  Long running sessions will be split into 12 hour segments to improve the ability to locate session segments of interest.



- The legends below the timeline are toggle buttons which help you filter the data you want to view on the timeline.
- Drill down to a Page or an AJAX call from the data displayed below the timeline to view details of the page or the AJAX call.

Monitor Server Request Performance

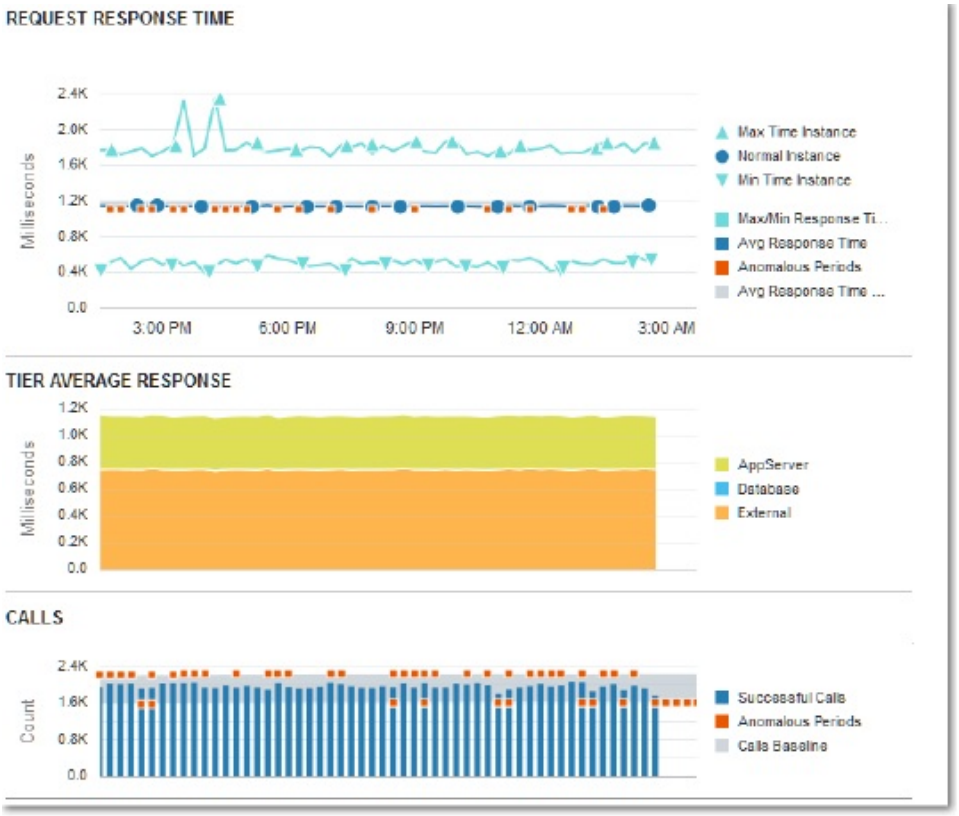
You must monitor HTTP requests to see how your application behaves as load varies. Clients, such as browsers used by your application users, perform various operations on your application resources and objects. Using Oracle Application Performance Monitoring, you can assess server-side performance down to the operation and method level. You can view and monitor application requests linked across servers. This level of detail helps you understand how clients access your application, and you can take action to improve end-user experience.

In the Oracle Application Performance Monitoring home page, the **Server Request** tile displays the top five server requests. Click the server request that you want to view details about. If the server request you are looking for is not among the top five:

1. In the left pane, click **Server Requests** to see a list of server requests.
2. Click the name of a server request to view details of the server request.

   In the **Diagram** tab, see a pictorial representation of the calls made by the server request. You can see how many calls the request has received, how many internal and external calls it has made and if there are any errors in any of these calls. Hover over the objects in the diagram to view details in the **Tooltip** pane.
3. Click **Metrics** to view further details of the server request.

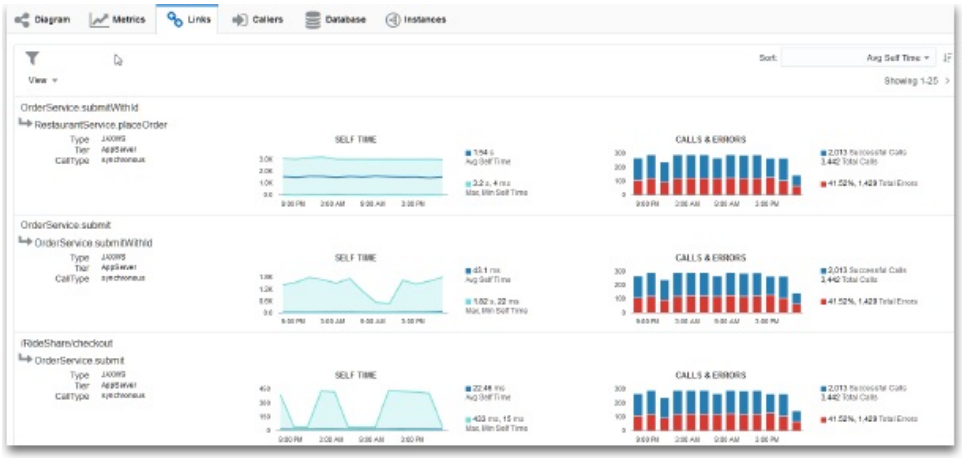
Use the Anomalies legend to turn on/off the information about anomalies and baselines. You can also turn off other elements like Maximum Time and Minimum Time to simplify your view.



- In the above example, the response time of the server request is highest at over 2 seconds, twice between 3 and 6 pm.
- The average response time of the external tier seems to be the highest among the three tiers.
- Check the Anomalous Periods and Average Response Time Baseline options to view the baseline and anomalies on request response time. The anomalies are depicted as spots above and below the baseline. You can also see the baseline and anomalies for Successful Calls.

4. Select the **Links** tab to view all the links related to the request.

   The Links tab links all application requests across servers. The tab also lists the callers of the server request, as well as the operation invocations made within the server request. If the server request makes external calls to other server requests, the user can click the other server request to navigate to its details.



View Metrics for a Group of Server Requests

You can view combined metrics for a group of related Server Requests.

Related Server Requests are grouped together into **Logical Server Requests** or **LSRs**, and you can view metrics for all these server requests together. LSRs are based on a unique combination of these 3 values — Server request start operation time, Deployment Time, and Genre Type. LSRs are available on applications created with the criterion AppServers with Classifications.

1. In the left navigation pane, select **Applications** to view a list of Applications.

   For each application listed, you can see the criteria of the application definition.

2. Click the name of the application you want to access or view more details for. The application details page is displayed.

   If this application was defined by **AppServers with Classifications**, you can view its LSR metrics.

3. In the left navigation pane, select **Logical Server Requests**. All the Server Requests based on your definition are listed here, irrespective of applications.

4. Click the name of the LSR you want to view details for.

   The **Metrics** tab displays details like Request Response Time, Tier Average Response and Call details. The **Server Requests** tab lists all associated server requests. You can drill down to view further details of the server requests.

Monitor All Objects Related to an Application

You can monitor all objects related to an application from the Applications page.

Define APM Applications

With Applications, you can define and save a filter to pull together a set of pages or server requests in the application, that match the defined filter criteria at a particular time point.

An administrator can create an Application Definition for pages with a specific URL, or server request with a specific name, deployment or application server. To create an Application Definition:

1. In the left navigation pane, select **Application Definitions** to view the Application Definitions screen.
2. Click **Create Application Definition**.
3. In the Create Application Definition screen, provide a name for the application definition.
4. Choose the object on which to base your application definition. The application definition can be a collection of Pages, Server Requests, Mobile Clients, or Application Servers with Classifications.
5. Click **Select Criteria** to specify the criteria for the selected object.
   - Pages – Specify the contents of your page URL.
   - Server Requests – Specify the name of the application server host, the deployment or the server request.
   - Mobile Clients — Specify the name of the mobile client, or the version of the mobile client.
   - AppServers with Classifications — Specify the classification based on which the AppServers will be filtered. If you want to see database information for an application, choose this option.

     The classifications that are available have been added by your administrator. See Adding Application Classifications in *Installing and Configuring Application Performance Monitoring* and talk to your administrator to get classifications added.

   All objects that meet these criteria will be part of the application definition. All entities that are related to the filtered objects are also part of the application. For example, if a Page is part of the application definition, all related AJAX calls and Application Servers are part of the application definition even though they are not part of the filter you defined.
6. Click **Apply**. Add one or more criteria as required.
7. Click **Save**.

   Soon after creating an application, the **Last Evaluated** field will temporarily display *Pending*. Refresh the page (*F5*) to monitor when it changes from *Pending* to an actual date/time when the application was evaluated.

The new application definition is listed in the Application Definitions page. You can select an application as a filter in the Home page, Page List page, Application Server List page, and Server Request List page. This will narrow the page focus to just the elements that are included in the scope of the Application Definition. You can delete an Application Definition by clicking the X icon in the respective row.

Use APM Applications

You can access an application to filter a set of pages or server requests in the application, that match the particular filter criteria defined in the application. All the related objects are part of the application definition too.

To use an Application:

1. In the left navigation pane, select **Applications** to view a list of Applications.

For each application listed, you can see the criteria of the application definition. An overall status of the application can be seen through information about End Users, Server Requests and the database tier.

2. Click the name of the application you want to access or view more details for.

   The application's home page displays detailed information about the application and related entities. You can see widgets for APM Alerts, Infrastructure Health (if Infrastructure Monitoring is enabled), and detailed information about associated entities in various tabs. All entities that are related to the filtered objects are also part of the application. For example, if a Page is part of the application definition, all related AJAX calls and Application Servers are part of the application definition even though they are not part of the filter you defined.

   - Click the number of Alerts to view the list of open alerts on the application.
   - The Infrastructure Health widget gives you an overall picture of the infrastructure.
   - View details of browser and application performance in the **Application Metrics** tab.
   - In the **Performance Analysis** tab, see how the server requests are distributed across application servers through the graph.
   - In the **End Users** tab, see the geo-map for all the pages in the application.
   - Click the Topology icon on top left corner of the pane to view/hide the application topology.

Viewing Flow Topology of an Application

Application Flow Topology allows you to see how the traffic of an application flows across various objects. You can see how the objects relate to each other based on the call flows that Oracle Application Performance Monitoring monitors for the selected application.

View relationships between *browser clients-application servers-databases*, or between *pages-AJAX calls- server requests-databases*. You can also view aggregated metrics for average response time, calls, and errors.

For example, the *AppServer* layer shows how calls flow from browser clients (pages and/or AJAX calls) to appservers, between appservers themselves, and to databases. The *Server Request* layer shows more details like call flow from specific pages to specific server requests.

To view the topology of an application:

1. In the left navigation pane, select **Applications** to view a list of Applications.
2. Select the application and in the left navigation pane, select **Flow Topology**.

   The topology diagram provides a quick overview of the performance of the application and all the objects interacting with the application.
3. Hover over the objects to view further details.
4. Select an object in the diagram to highlight the path of the object. The details of the selected object are listed below the diagram.
5. Select different **Diagram Controls** to view other aspects of the topology.

Monitoring End User Experience of a Web Application

You can monitor all objects related to a web application from the Web Applications page.

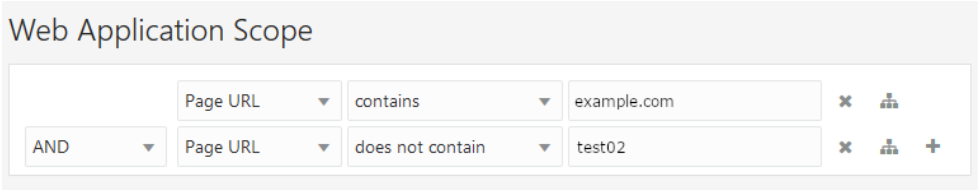
Define APM Web Applications

With Web Applications, you can define and save a filter to pull together a set of pages in the application, that match the defined filter criteria at a particular time point.

An administrator can create a Web Application Definition for pages with a specific URL or name. To create a Web Application Definition:

1. From the main menu, select **Monitoring**, then click **Monitoring Admin**.
2. Click **APM Web Application Definition**.
3. Click **Create**.
4. In the Create Web Application screen, provide a name for the web application definition.
5. Choose the scope on which to base your web application definition. The web application scope can be a Page URL or OMC Tag. Various operators are available such as: "contains", "starts with", etc. For more complex rules, you can extend the scope matching several rules in a logical tree.

Figure 3-1 Web Application Scope



In this example, the web application definition is looking for page URLs that contain example.com and does not contain test02.

The new web application definition is listed in the Web Application Definitions page. You can select a web application as a filter in the Home page, Page List page, and Application Server List page. This will narrow the page focus to just the elements that are included in the scope of the Web Application Definition. You can delete a Web Application Definition by clicking the X icon in the respective row.

Additional Reporting Classification

Configure the URL and context values from the traffic collected from pages, page updates, and Ajax calls for your Web Application definition.

Data can be extracted from the input source and optionally manipulated. Input sources are typically:

- **Base URL**

- **URL Argument**
- **Page Title**
- **Attribute 1**
- **Attribute 2**

You can manipulate the output by selecting **Use Regular Expressions**. After selecting the input source, add a search pattern and type what it's going to be replaced by. Test the regular expression by typing an example in the **Test On** text field and click **Run**.

In addition to the custom regular expressions defined by the user, Oracle Application Performance Monitoring also provides **Always Applied Output Manipulation** to the definition.

The manipulations include the following:

- Removal of date strings from URLs, page titles, and attributes
- Removal of time strings from URLs, page titles, and attributes
- Removal of email addresses from page titles and attributes

Test Configuration

Click **Collect Sample Data** to collect a 15 minute sample of live traffic. Once the sample has gathered data, you can run the definition and view how data will be manipulated by this web application.

Use APM Web Applications

You can define a web application to match a set of pages to a set filter criteria.

Specific web application metrics like sessions, pages, and AJAX calls can be seen from the APM UI in context of the web application entity. Graphs related to the web applications metrics can be seen from the Monitoring UI in context of the web application entity. For quick access to the data from web applications, select a web application via a global filter, which sets the context for further navigation in the APM or Monitoring UI.

View Web Application Metrics

From the Web Application metric page, you can access diagnostic data, Ajax calls, and Sessions in more detail.

1. From the Monitoring menu, click **Entities**.
2. Filter by **APM Web Application** Entity Types.
3. Click on the **APM Web Application** Entity.

Click the **Actions** button to monitor Pages, Ajax calls, and Sessions related to the Web Application metric.

View Web Application Pages

Visualize page data related to the Web Application using Oracle Application Performance Monitoring.

From the Web Application Pages UI, you can use different filters from the drop-down menu or use quick filters such as:

- **Load**: This will show server requests that helped construct the page.
- **Partial**: This will show end user interactions that triggered some update to the page through Ajax calls.
- **Full**: This will show pages that modified the URL due to user activity, meaning that the updated pages can be bookmarked and are accessible in browser history.

Partial and **Full** page loads will be seen in the context of Single Page Applications (SPA), where Ajax calls are executed to update page content instead of loading a complete new page. These filters will offer links to the relevant server requests.

View Detailed Information about a Server Request Instance

If you have deployed APM Java Agent, you can view detailed information about a Server Request Instance.

Viewing Detailed Information about a Server Request Instance

You can view detailed information of a server request instance to get an insight into performance of major operations. You can view information like usage of CPU and Heap resources, memory allocation and thread stack that will help you analyze a server request instance. To view detailed information of a server request instance:

1. In the Server Request Instance list page, view the summary for an instance. You can view details like CPU usage, memory allocation and the total number of snapshots taken.
2. Click the Instance that needs attention.

   The Instance details page is displayed.
3. In the **Call Tree** tab, click  to view a snapshot of the instance.
   - The Snapshot pane indicates when the first snapshot was taken, frequency of the snapshot, and the total number of snapshots.
   - Details about the instance like memory allocation, CPU usage, Block Time and garbage collection overhead are also displayed.

   Scroll through the pane to view other details. This pane displays all the stacks for the specific operation, aggregated as a tree.
4. Click the **Snapshots Timeline** tab to view a chart of the snapshots across time. Click an item on the chart to view the stack information window. Each snapshot is represented by a circle on the chart. The status of the snapshot is depicted by the color of the circle.

Viewing Error Details for a Server Request Instance

If a Server Request Instance is indicating failure, you can view the details of the error in the Error Details pane. To view details of an error in a server request instance:

1. In the Server Request Instance list page, look for instances with status as Failure, indicated by a red X in the Status column,  .

2. Optionally, click the Server Request Instance with the error to view the Server Request Instance details page.

3. The Failure icon displays an additional badge for Error Context,  . Click to view details of the error in the Error Details pane.

   The Error Details pane displays error messages, codes and error stack.

> **Note:**
>
> Detailed information for a server request instance and the Error Details pane are available only if you have deployed the APM Java Agent or the APM .Net Agent, and is not available with other APM Agents.

Monitor the Performance of an Application Server

Oracle Application Performance Monitoring lets you view details about an Application Server.

In the Oracle Application Performance Monitoring Home Page, the **AppServers** tile displays the top five Application Servers. Click the Application Server that you want to view the details for. If the Application Server you are looking for is not among the top five:

1. In the left navigation pane, click **AppServers** to see a list of AppServers.

   All the Application Servers that are currently running are displayed here.

2. Click the Application Server for which to view the details.

   You can view graphs of memory and CPU usage, garbage collection, response time, load and thread pool.

3. Click the **Metrics** tab to view metrics of the application server over a selected time range.

4. Click the **Server Requests** to view a list of server requests being handled by the application server. Click a server request to view its summary.

5. Click the **Database** tab to view details of the SQL calls.

6. Click **Thread Profiler** to view a list of all thread profilers belonging to this application server. You can click on a thread profiler to view its summary.

7. In the quick menu on the top right corner, click **Start Thread Profiler** to capture thread data (like stack snapshots, thread memory allocation, CPU consumption, blocked time) for a small duration of time that you can specify.

8. From the quick menu, click **View Log** to view log events for the application server.

   The Log Visual Analyzer displays the error logs for the Application Server. You can change the time range and view data accordingly.

Collect Thread Profiler Data for an Application Server

You can collect detailed data that are not generally captured by Oracle Application Performance Monitoring through the Thread Profiler.

Thread Profiler can collect detailed data about an application server for a small duration of time. It can collect a variety of data like thread data (stack snapshots, thread memory allocation, CPU consumption, blocked time), Garbage Collection data, JVM parameters, and System Parameters. This data can be used to understand JVM activities or threads based on CPU or Memory consumption.
You can have a maximum of 5 Thread Profiler schedules defined for an APM agent at a time. In case you already have five thread profiler schedules, ensure you delete an old schedule before creating another one.

To define a thread profiler schedule:

1. From the AppServer details page of the required Application Server, click the quick menu icon, and select **Start Thread Profiler**.

   OR

   From the left navigation menu, select **Diagnostic Snapshots**, select **Thread Profiles** and click **Start Thread Profiler**.
2. In the Thread Profiler dialog box, specify the details:
   - Specify a **Name** for the thread profiler, if you want to change the default name.
   - Select an **AppServer** if it is not already selected.
   - Specify a **Duration** for the profiler. You can choose up to 5 minutes.
   - Optionally, provide a **Description** for the profiler.
3. Click **Start Profiler**.

   You can view this Thread Profiler from the **Thread Profiles** page. You can also view the thread profile **View Diagnostic Snapshots** option from the quick menu in the respective AppServer details page.

Collect JFR Data for an Application Server

Your Oracle Application Performance Monitoring Java agent can collect diagnostic data using the Java Flight Recorder (JFR).

Prerequisites:

1. JFR dump is supported only on HotSpot JDKs (Sun, JRockit, Oracle JDKs), and not on IBM JDK.
2. JFR dumps can be analyzed through JMC (Java Mission Control). For Oracle JDK 1.7.0_04 and later (but earlier than JDK 1.8.0_40), JVM process should be run with the following Java options:

   ```
   -XX:+UnlockCommercialFeatures -XX:+FlightRecorder
   ```

   These Java options are not required for JDK 1.8.0_40 and later.
3. Before you can collect JFR data, ensure that the JFR is running.

To collect JFR data:

1. From the AppServer details page of the required Application Server, click the quick menu icon, and select **Dump JFR**.

   OR

   From the left navigation menu, select **Diagnostic Snapshots**, select **JFR Dumps** and click **Dump JFR.**

2. In the Dump JFR dialog box, specify the details:
   - Specify a **Name** for the JFR dump, if you want to change the default name, which is a time stamp.
   - Optionally, click **Add Description** and provide a description for the JFR dump.
   - Select an **AppServer** if it is not already selected.
3. Click **Dump JFR**.

   The JFR dump collection starts in the next harvesting cycle of the APM agent (in about a minute), and it takes a few minutes to perform this operation.

   You can view the JFR Dump from the **JFR Dumps** page. You can also view the JFR dump from the **View Diagnostic Snapshots** option from the quick menu in the respective AppServer details page.

Viewing and Downloading JFR Dump

To view and download JFR dump:

1. From the left navigation menu, select **Diagnostic Snapshots**, and select **JFR Dumps.**
2. To download the `.jfr` file, click the download button at the right end of the respective JFR Dump row.
3. To view the JFR Summary, click the name of the JFR Dump. The JFR Summary page is displayed.
4. In the JFR Summary page, click the download button to download the JFR Report.

Disabling JFR Dump

To disable JFR dump:

1. Navigate to the folder `apmagent/config/<AppServer Name>`, and edit the `Properties.json` file.
2. Set the static property `oracle.apmaas.agent.deepdive.disableJFR` to `true`. To enable, set the property to `false`.
3. Restart the application server.

Collect Class Histogram for an Application Server

Your Oracle Application Performance Monitoring Java agent can collect Class Histogram data for analysis of the performance of your application server.

Prerequisites:

1. Class Histogram is supported only on Sun and JRockit JDKs.

2. Ensure that `tools.jar` is set in the Java classpath.

To collect Class Histogram for an application server:

1. From the AppServer details page of the required Application Server, click the quick menu icon, and select **Class Histogram**.

   OR

   From the left navigation menu, select **Diagnostic Snapshots**, select **Class Histograms** and click **Create Class Histogram.**

2. In the **Create Class Histogram** screen, specify the details:
   - Specify a **Name** for the Class Histogram if you want to change the default name, which is a time stamp.
   - Optionally, click **Add Description** and provide a description for the Class Histogram.
   - Select an **AppServer** if it is not already selected.
3. Click **Create Class Histogram**.

   The Class Histogram is created in the next harvesting cycle of the APM agent (in about a minute), and it takes a few minutes to perform this operation. Once the Class Histogram is created, it is listed in the Class Histograms page, with the class count details. Click the name of the Class Histogram to view the summary.

   You can view the Class Histogram from the **Class Histograms** page, and also from the quick menu in the respective AppServer details page.

Disabling Class Histogram

To disable Class Histogram:

1. Navigate to the folder `apmagent/config/<AppServer Name>`, and edit the `Properties.json` file.
2. Set the static property `oracle.apmaas.agent.deepdive.disableClassHistogram` to `true`. To enable, set the property to `false`.
3. Restart the application server.

Collect Heap Dump for an Application Server

Your Oracle Application Performance Monitoring Java agent can collect Heap Dump data for analysis of the performance of your application server.

Prerequisites:

1. Heap Dump can be taken on Sun (V1.6 or later), JRockit and IBM (v1.6 and above) JDKs.
2. In case of IBM 1.6 JDK, set the environment variable `IBM_HEAPDUMPDIR` to the directory in which the Heap Dump has to be stored. If this variable is not set, the Heap Dump will be stored in the directory from where the application server is running.

To collect Heap Dump:

1. From the AppServer details page of the required Application Server, click the quick menu icon, and select **Take Heap Dump**.

OR

From the left navigation menu, select **Diagnostic Snapshots**, select **Heap Dumps** and click **Take Heap Dump.**

2. In the **Take Heap Dump** screen, specify the details:
   - Specify a **Name** for the Heap Dump if you want to change the default name, which is a time stamp.
   - Optionally, click **Add Description** and provide a description for the Heap Dump.
   - Select an **AppServer** if it is not already selected.
3. Click **Take Heap Dump**.

   The Heap Dump collection starts in the next harvesting cycle of the APM agent (in about a minute), and it takes a few minutes to perform this operation. The path where the heap dump file is stored is displayed after the heap dump is taken.

   You can view the Heap Dump from the **Heap Dumps** page, and also from the quick menu in the respective AppServer details page.

Disabling Heap Dump

To disable Heap Dump:

1. Navigate to the folder `apmagent/config/<AppServer Name>`, and edit the `Properties.json` file.
2. Set the static property `oracle.apmaas.agent.deepdive.disableHeapDump` to `true`. To enable, set the property to `false`.
3. Restart the application server.

Integrate Application Performance Monitoring Events with JFR

Application Performance Monitoring events are integrated with JFR by default.

APM Events are integrated with JFR using Java Mission Control (JMC). The data captured by the APM Agent can then be viewed on JMC, thereby providing more insight into all APM Events.

Prerequisites:

- Ensure that the JFR support is set to ON.
- A JFR recording should be running.
- In the `Properties.json` file located in the `apmagent/AdminServer` folder, ensure that the following property is set to `TRUE`:

  ```
  oracle.apmaas.agent.deepdive.jfrEvent
  ```

  By default, this property is set to `TRUE`, which means that the JFR- APM Event integration is enabled.

To view Application Performance Monitoring Events in JMC:

1. If the above prerequisites are met, once you deploy the APM Java Agent, you will see the complete list of APM Events when you start a recording.

You can enable all or specific events from the list. Follow JMC documentation for instructions on how to enable events.

2. Once you have a recording of the JFR, you can see the APM events that you have selected, and data about those events in the JMC UI.

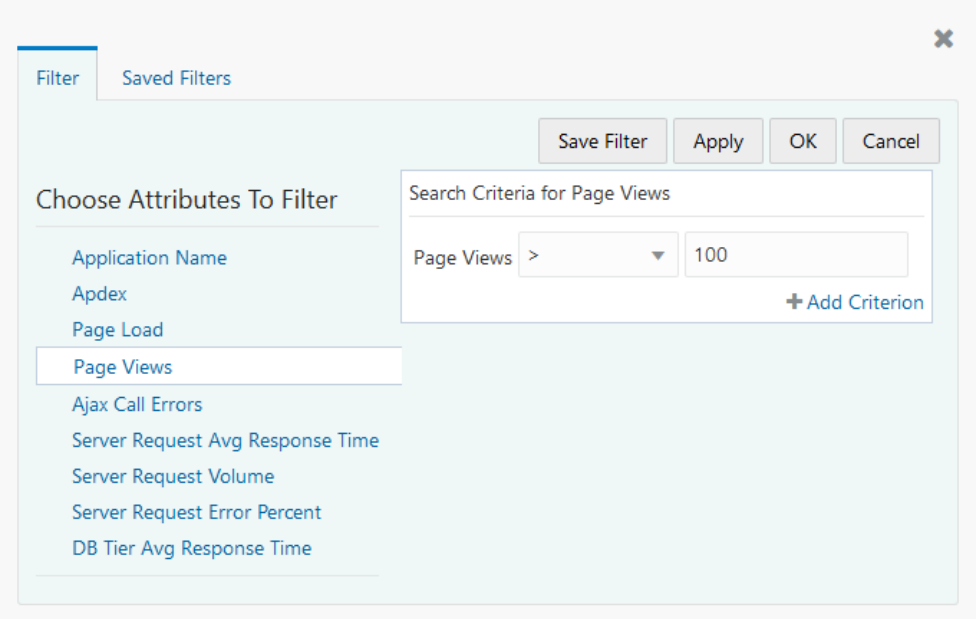
   Various tabs in JMC provide details on APM Events, and metrics about APM entities.

Create and Manage Filters

Filters are a way to display useful information for the task at hand in Oracle Application Performance Monitoring.

1. To create or manage a filter, click **Manage Filter**. 
2. In the Manage Filter screen, choose an attribute to filter and its search criteria.

Figure 3-2 Page Views Filter



In the manage filter screen, the filter is set to display pages that have more than 100 page views.

3. Click **Save Filter**. In the Save Filter screen, select a **Name**, **Description**, and click **Monitored Filter**.
   The filtered set of pages will show as aggregated metrics in the Web Application.

4

Administer Oracle Application Performance Monitoring

An Oracle Application Performance Monitoring administrator can deploy and administer the service in your environment.

Topics

Typical Tasks for Administering Oracle Application Performance Monitoring

Here are the administrative tasks for Oracle Application Performance Monitoring.

Task	Description	More Information
Deploy Oracle Application Performance Monitoring	Download and install Oracle Application Performance Monitoring	Deploy Oracle Application Performance Monitoring
Specify servlet configuration options	Specify how your web application / servlet is being monitored by Oracle Application Performance Monitoring.	Define Servlet Configuration Options
Set rules for alerts	Set rules for email alerts	Set Alert Rules
Set up Synthetic Monitoring	Set up synthetic monitoring, and monitor application performance through synthetic tests	Typical Workflow for Using Synthetic Monitoring
Set up End User Monitoring	Set up End User Monitoring with manual browser injection, enable and configure settings	Set Up End User Monitoring

Task	Description	More Information
Set up Synthetic Monitoring	Set up Synthetic Monitoring by simulating user paths to detect possible errors	Set Up Synthetic Monitoring
Set up Custom Instrumentation	Set up Custom Instrumentation and start monitoring technologies not supported out-of-the-box.	Set Up Custom Instrumentation
Enable Privacy Settings	Enable Privacy Settings to stop APM from storing personal identifiable information.	Enable Privacy Settings

Customize APDEX Settings

The Application Performance Index, APDEX is a measurement of overall performance of your application.

APDEX is a summary of various measurements, with appropriate weightage on certain important results. Outliers which are otherwise ignored are given due importance in this index. The formula used to calculate APDEX = (Number of Satisfactory samples + (Number of Tolerating samples) / 2) / Total samples.

Session Health uses APDEX results for page and AJAX requests, along with the impact of error executions of java scripts and AJAX requests. With the Session health settings, you can influence the weightage given to the various components while calculating session health.

You can customize settings for the Application Performance Index. To customize settings for APDEX:

1. In the left navigation pane, click **Administration** and select **Metric Settings**.
2. In the APDEX Settings pane, provide **Satisfactory** and **Tolerable** values for Page Load Time and Ajax Call Response Time.
3. In the Session Health Settings pane, provide a weightage for each element of the formula used to calculate session health:
    a. Page Performance Weight
    b. Ajax Performance Weight
    c. Javascript Errors Weight
    d. Ajax Errors Weight
4. Click **Save** on the top right corner of the screen to save your settings.

The new settings will be used to determine APDEX value and to rate the session health. Note that elements for which there is no data (*Example:* No page views) will not be considered in the formula.

Associate Entities Using Tags

You can associate Application Servers to Database using tags.

To associate an application server to a database:

1. From the Oracle Management Cloud left navigation menu, click **Administration** and select **Entities Configuration**.

2. Select **Tags** to view a list of tags that have already been discovered.
3. In the Tags screen, click **New Key** to create a new association with tags.
4. In the **Create New Key** screen, provide a name and value for the key.

Deleting Tags

1. From the Oracle Management Cloud left navigation menu, click **Administration** and select **Entities Configuration**.
2. Select **Tags** to view a list of tags that have already been discovered.
3. In the Tags screen, search for the tag using the search box.
4. Select the tag, and click the Delete Tag button.
5. Click **OK** to confirm the operation.

   The selected tag is deleted.

Associate Application Servers to a Database Automatically

Application Performance Monitoring automatically creates an association between the App Server and the Database. When the association is not automatic, you can dynamically make the association through matching association tags.

Associate Application Servers to a Database through tags for a new Target

After installing and deploying Application Performance Monitoring, you can associate application servers to a database through tags. The below procedure is for a fresh installation of Application Performance Monitoring.

To associate an APM-Invisible Database with Application Server using tags:

1. Download the APM Agent Software Installer and provision APM Agent. See Types of APM Agents to install the correct APM Agent.
2. Shut down the application server to which you are associating the database.
3. Add the below property in the Properties file to generate an association from the application server to the database:

   For APM Java agent:

   a. From the `apmagent/config` folder, edit the `AgentStartup.properties` file and add the below property:

   ```
   <KeyName>=<Tag Value>
   oracle.apmaas.agent.appServer.uses = Tag_Weblogic_To_Oracle
   ```

   b. **Generic property tag**: In the `AgentStartup.properties` file, you can add a new list of tags in the format `tag-name [ = tag-value ]` to the below property:

   ```
   oracle.apmaas.agent.appServer.tags
   ```

Example:

```
oracle.apmaas.agent.appServer.tags=credToUse=exampletag1,exampletag2,tag3=exampletag3
```

The above property will create 3 tags:

- `credToUse=exampletag1`
- `exampletag2`
- `tag3=exampletag3`

For APM .Net agent:

- From the `<Path where APM Agent Software is extracted>\Oracle APM .NET Agent` folder, edit the `AgentConfig.ini` file and add the below property:

```
<KeyName>=<Tag Value>
oracle.apmaas.agent.appServer.uses = Tag_DotNet_To_Oracle
```

In the above configuration file, the following is the Source Entity Marker tag:

```
assoc_source:<AssociationHint>=<AssociationTag>
```

Where:

- `assoc_source:<AssociationHint>` is the tag key.

  `<AssociationHint>` can be any free form text used to identify, and therefore associate the set of destination entities. The `<AssociationHint>` used for the source entity, should match the one used for the destination entity(entities).

- `<AssociationTag>` is the tag value.

  `<AssociationTag>` is the intended association tag. It is optional, and if a value is not provided, it defaults to `omc_uses` association tag.

An entity can be tagged with one or more source marker tags if the source entity can have different associations with different sets of destination entities. Ensure that the source marker tag is unique.

4. Start the application server.
5. Verify that the tags are added to the application.
   a. From the Oracle Management Cloud left navigation menu, click **Administration** and select **Entities Configuration**.
   b. Select **Tags** to view a list of tags that have already been discovered.
6. Verify that the application server is associated with the database in any of these ways:
   - From the Application Definition screen, view the topology.
   - From the App Server screen, click the Topology button to see the association.

You can also associate an App Server to a database from the APM UI. See Associating Entities Using Tags.

Discovering App Servers and Database entity using tag from Monitoring Service for a New Installation

1. Install the Oracle Cloud Agent.
2. To discover the database, modify the `db.json` and add the tag element.

   Example:

```
{ "entities":[
      {
         "name":"Oracle_DB",
         "type":"omc_oracle_db",
         "displayName":"Oracle_DB",
         "timezoneRegion":"PDT",
         "credentialRefs":["SQLCreds"],
         "properties":{
                          "host_name":
{"displayName":"MachineName","value":"sample.test.com"},
                          "port":
{"displayName":"OraclePort","value":"1000"},
         "protocol":{"displayName":"Protocol","value":"TCP"},
                          "sid":
{"displayName":"SID","value":"sample"},
         "capability": {"displayName": "capability","value":
"monitoring"}
                       },
"tags": {
       "assoc_dest:<assoc_name>" : ""
   }
        }
]}
```

   Where <assoc_name> can be any string based on user preference. Discover the database entity using cloud agent (`omcli add_entity agent db.json`).

3. To discover the application server, modify the app server json and add the tag element.

   Example:

```
{
"entities":[
       {
         "name":"Tomcat_Sample_Demo8",
        "tags": {
                "assoc_source:<assoc_name>" : ""
                },
         "type":"omc_tomcat",
         "displayName":"Tomcat_Sample_Demo8",
         "timezoneRegion":"PDT",
         "properties":{
        "host_name":{
            "displayName":"Host",
                          "value":"abcd.test.com"
```

```
                        },
                    "jmx_port":{
                "displayName":"JMX Port Number",
                "value":"9999"},
            "jmx_user_name":{
                "displayName":"JMX User Name",
                "value":""},
            "jmx_password":{
                "displayName":"JMX Password",
                "value":""},
            "jmx_protocol":{
                "displayName":"Communication Protocol",
                "value":"rmi"},
            "jmx_service":{
                "displayName":"Service Name",
                "value":"jmxrmi"},
            "ssl_trust_store":{
                "displayName":"SSL Trust Store (required when SSL is
enabled)",
                "value":""},
            "ssl_trust_store_password":{
                "displayName":"SSL Trust Store JMXPassword (required
when SSL is enabled)",
                "value":""},
            "ssl_trust_store_password":{
                "displayName":"SSL Trust Store JMXPassword (required
when SSL is enabled)",
                "value":""},"version":{
                "displayName":"Apache Tomcat Version",
                "value":"8"},
            "catalina_base_directory_path":{
                "displayName":"Catalina Base Directory Path",
                "value":"/scratch/opt/apache-tomcat-8.0.29"},
            "LoggingConfigurationFilePath":{
                "displayName":"Logging Configuration File Path",
                "value":""},
            "LogLocationCatalina":{
                "displayName":"Log Location Catalina",
                "value":""},
            "LogLocationLocalhost":{
                "displayName":"Log Location Local Host",
                "value":""},
            "LogLocationHostManager":{
                "displayName":"Log Location Manager",
                "value":""},
            "LogLocationHostManager":{
                "displayName":"Log Location Host Manager",
                "value":""},
            "capability": {
              "displayName": "capability",
              "value": "monitoring"}

              }
            }
```

```
]
}
```

where <assoc_name> string should be same as in the database json in step 2. Discover the Tomcat entity using cloud agent (`omcli add_entity agent tomcat.json`.)

4. "omc_uses" association should be automatically created between Tomcat and the database entity.

Update the App Servers and Database entity using tag from monitoring Service

You can add new tags to existing entities, but cannot update existing tags from the cloud agent `omcli cmd`. Here is a sample json:

```
{ "entities":[
      {
         "name":"Sample_DB",
         "type":"omc_sample_db",
         "displayName":"Sample_DB",
         "timezoneRegion":"PDT",
         "credentialRefs":["SQLCreds"],
         "properties":{
                          "host_name":
{"displayName":"MachineName","value":"sample.test.com"},
                          "port":
{"displayName":"OraclePort","value":"1111"},
         "protocol":{"displayName":"Protocol","value":"TCP"},
                          "sid":{"displayName":"SID","value":"orcl12c"},
         "capability": {"displayName": "capability","value":
"monitoring"}
                       },
"tags": {
       "assoc_dest:<assoc_name>" : "",
       "assoc_dest:<assoc_name2>" : "",

       "key1" : "value1"

   }
        }
]}
```

Use the `omcli update_entity agent db.json` command to upload the new tags from the entity json to Oracle Management Cloud.

Define Locations for Synthetic Tests

An administrator defines locations from where synthetic tests will be run.

Prerequisites before you can define locations for synthetic tests:

- Install Firefox browser on the machine where the cloud agent is installed:
  - Oracle Linux 6: Firefox version 45
  - Oracle Linux 7: Firefox version 61-66

> **Note:**
>
> Firefox is the only supported browser. Other Firefox versions including Beta versions are unsupported.

You can check if Firefox is present on the cloud agent machine by running the command `firefox --version`.

Install the relevant Firefox version on the host, and specify the correct path of the Firefox executable while defining a location. To migrate existing tests from Firefox 45 to Firefox 61, see Migrating Synthetic Tests to Firefox 61.

- Firefox should be part of the `PATH` shell variable of the user used to run the cloud agent. For example, if Firefox executable is available at `/scratch/firefox`, then `/scratch/firefox` should be part of the `PATH` variable.
- Install a Cloud Agent on the machine where the tests will be run. See *Deploying Cloud Agents* in *Deploying and Managing Oracle Management Cloud Agents* for instructions.
- Ensure Perl is installed on the machine running the Cloud Agent.
- Create an X Server pool, a set of X Servers which will be used to execute the Selenium test.

  You can check if an X Server pool exists by running the command `ps –ef`. For example, to check if there are any Xvfb X servers running, run following command:

  ```
  ps -ef | grep Xvfb
  ```

- Any X Server can be used, but the recommended X Server is Xvfb. To create an X Server pool using Xvfb, run the following commands:

  ```
  Xvfb :1
  Xvfb :2
  Xvfb :3
  Xvfb :4
  Xvfb :5
  ```

  Every time the above commands are run, an X Server with the specified display port is created. These display ports can be used to specify the X Server pools, while creating a location.

> **Note:**
>
> In case of an error when you run the above command, run this command instead:
>
> ```
> Xvfb :1 -nolisten inet6
> ```

The following is an example on how to setup Xvfb server and auto-start at system boot:

1. `$ sudo mkdir -p /usr/local/sbin # (if not already present)`
2. `$ sudo cp xvfb_start.sh xvfb_stop.sh /usr/local/sbin`
3. `$ sudo chmod 744 /usr/local/sbin/xvfb_start.sh /usr/local/sbin/xvfb_stop.sh`
4. `$ sudo cp etc_systemd_system_oracle.service /etc/systemd/system/xvfb.service`
5. `$ sudo systemctl daemon-reload`
6. `$ sudo systemctl enable xvfb`
7. `$ sudo systemctl start xvfb`
8. `$ sudo systemctl status xvfb`

- In the file `/etc/hosts`, replace

  `127.0.0.1  localhost.localdomain  loghost localhost`

  with

  `127.0.0.1  localhost   localhost.localdomain  loghost.`

- The cloud agent should not be installed in a symbolic link directory, when installing the cloud agent the value provided for `AGENT_BASE_DIRECTORY` parameter should not be a symbolic link.

To define locations:

1. In the left navigation pane, click **Administration**.
2. Select **Locations** to view a list of locations that are available.
3. On the Locations screen, click **Add Location**.
4. In the Add Location screen, specify information about the new location:
   - Name: Specify a name for the location
   - Available Agents: Select a cloud agent for the location.
   - Specify other optional details for the location — Longitude, Latitude, City and Country.
   - Optionally specify Proxy details for the location — Host, Port, Username and Password.
   - Capacity: Specify the maximum number of machines for this location.
   - X Server Pool: Specify the X Server pool to be used while executing the Selenium test. If you have X servers running on ports, you can specify the port number.

     Example: If X servers are running on display ports 1, 2 and 3, specify 1-3 as value for this property.
   - Firefox Path: Specify the path of Firefox executable. Example: `/usr/bin/firefox`.
5. Click **Save**.

The new location will be displayed in the list of Locations, and will be available for selection while creating a synthetic test.

Migrating Synthetic Tests to Firefox 61

1. Install version 61 (or higher) of Firefox on the machine where you will install the Cloud Agent.
2. On the same machine, install Cloud Agent version 1.33 or higher.
3. Define a Location using this Cloud Agent.
4. Once this Location is active in the system, edit your synthetic test/s and add this new location to the test/s.
5. Wait for Synthetic Tests to run on this new Location. You can verify this in the Instances tab where you will see data coming from two Locations. You can then edit the test and delete the location which was created using an older version of Firefox.

Configure Errors and Error Messages

APM Java Agent can automatically detect operation invocation errors while monitoring application flows. An invocation error could be defined as an unexpected event happening as an operation is invoked. To detect such events and classify them as errors, APM agents rely on a set of default rules:

- HTTP response status code that ranges from 400 to 599
- Exceptions thrown during operations execution
- SOAP message response with a Fault block
- OSB error codes

In some cases, however, you may want to ignore an error and have the invocation classified as success. For example, based on your Application behavior, you might want to treat HTTP 403 response codes as success. With Error configuration, you can modify the default error classification rules, customizing them using an opt-out / opt-in mechanism, excluding or including your own defined rules.

To configure errors:

1. Open the file `Error.json` in an editor from the following path:
   - To configure errors on all the APM Agents in a domain, edit this file - `<DOMAIN_HOME>/apmagent/config/Error.json`.
   - To configure errors on the APM Agent on a specific server, edit this file - `<DOMAIN_HOME>/apmagent/config/<server-name>/Error.json`.
2. Delete the content of the `Error.json` file. Replace it with the following lines, modifying to specify your error rules:

```
{
  "type" : "error",
  "configurationsPerComponentType" :
  [
    {
      "componentTypes" : [],
      "excludeReturnCodes" : {},
      "excludeReturnCodesRegex" : {},
```

```
      "excludeErrors" : {},
      "excludeErrorMessages" : {},
      "includeReturnCodes" : {},
      "includeReturnCodesRegex" : {},
      "includeErrors" : {},
      "includeErrorMessages" : {},
      "includeFirst" : false
    }
  ]
}
```

3. Save and close the `Error.json` file. You do not have to restart the application server for these rules to take effect.

   Example: Here's an example where a HTTP error 403 is converted from an error to an acceptable event.

```
{
  "type" : "error",
  "configurationsPerComponentType" :
  [
    {
      "componentTypes" : ["SERVLET"],
      "excludeReturnCodes" : {"HTTP 403" : []},
      ...
      "includeFirst" : false
    }
  ]
}
```

You can use `configurationsPerComponentType` to define the error rules. Each rule (exclude/include) is expressed as a key-value pair, where the key defines the error code selection and value defines the list of operations the rule applies to. The value is optional and if not specified, the error code selection applies to all operations of the specified `componentTypes`. Here are the rules you can use to configure the errors:

Rule Type	Property	Type	Description
Component Type	`componentTypes`	Array	List of component types the configuration should be applied for. The same configuration can be applied to multiple types.
Opt-out (Exclude Rules)	`excludeReturnCodes`	Map	Key - an error code in String Value - an array of operation names in String
	`excludeReturnCodesRegex`	Map	Key - Regex pattern for the error code Value - a list of regex patterns for operation names

Rule Type	Property	Type	Description
	excludeErrors	Map	Key - a string of error type. Error type key has to be exactly the same as the error type shown in Error Detail page. Value - a list of regex patterns for operation names
	excludeErrorMessages	Map	Key - a regex pattern of an error message Value - a list of regex patterns for operation names
Opt-in (Include Rules)	includeReturnCodes	Map	Key - error code in String Value - an array of operation names in String
	includeReturnCodesRegex	Map	Key - Regex pattern for the error code Value - a list of regex patterns for operation names
	includeErrors	Map	Key - a string of error type. Error type key has to be exactly the same as the error type shown in Error Detail page. Value - a list of regex patterns for operation names
	includeErrorMessages	Map	Key - a regex pattern of an error message Value - a list of regex patterns for operation names
Include First	includeFirst	boolean	Determines which set of properties the configuration apply first to exclude or include server request instances from being marked as fault.

Notes on how these error rules work:

- **includeFirst**
  - If includeFirst is false, opt-out rules are applied first in the following order: excludeReturnCodes → excludeReturnCodesRegex → excludeThrowableClasses →

excludeThrowableMessages → includeReturnCodes → includeHttpReturnCodesRegex → includeThrowableClasses → includeThrowableMessages

- If `includeFirst` is true, opt-in rules are applied first:: includeReturnCodes → includeHttpReturnCodesRegex → includeThrowableClasses → includeThrowableMessages → excludeReturnCodes → excludeReturnCodesRegex → excludeThrowableClasses → excludeThrowableMessages

- Opt-in and Opt-out properties are independent of each other. A server request needs to satisfy only one of those properties in the configuration.
- **All operations:**
  For opt-in and opt-out properties, specify an empty list value or a list with an empty string to exclude or include any operation name with a key. For example, if you want agent to not report any error with HTTP 500 return code regardless of its operation name, you would define the following:

```
excludeReturnCodes : {"HTTP 500" : []}
or,
excludeReturnCodes : {"HTTP 500" : [""]}
```

- Look for these values in these APM views:

Server Request Instance page, Error Details tab	Operation name
Server Request Instance page, Call Tree table	– Operation name – Component Type
Error Details pane	– Error – Error code – Error message

- **Using Opt-in Properties:** Use Opt-in properties if you use regex to opt-out many errors from the default rules but want to allow some cases to be still treated as errors. For example, if you want to exclude HTTP 5XX from the rules except HTTP 500, set the value of `includeFirst` to `true` so that opt-in properties can be applied first in the configuration.

```
{
  "type" : "error",
  "configurationsPerComponentType" :
  [
    {
      "componentTypes" : ["SERVLET"],
      "excludeReturnCodes" : {},
      "excludeReturnCodesRegex" : {"HTTP 5[0-9][0-9]" : []},
      "excludeErrors" : {},
      "excludeErrorMessages" : {},
      "includeReturnCodes" : {"HTTP 500" : ["/hello/errorConfigTest/
throwException (GET)", "/hello/errorConfigTest/throwException
```

```
(POST)"]},
      "includeReturnCodesRegex" : {},
      "includeErrors" : {},
      "includeErrorMessages" : {},
      "includeFirst" : true
    }
  ]
}
```

- **Two Types with Overlapping Properties**
  Two or more component types can share the same customized rules if you specify all the types:

```
{
  "type" : "error",
  "configurationsPerComponentType" :
  [
    {
      "componentTypes" : ["SERVLET", "JAXRS", "SOA"],
      "excludeReturnCodes" : {},
      "excludeReturnCodesRegex" : {},
      "excludeErrors" : {"java.lang.RuntimeException" : ["/hello/
errorConfigTest/throwException .*"]},
      "excludeErrorMessages" : {},
      "includeReturnCodes" : {},
      "includeReturnCodesRegex" : {},
      "includeErrors" : {},
      "includeErrorMessages" : {},
      "includeFirst" : false
    }
  ]
}
```

- **One Type with Multiple Configurations**
  One type can belong to more than one `configurationsPerComponentType` if properties in `configurationsPerComponentType` do not overlap. This is useful if one type share the same properties as other component types but have one or two property that don't. Make sure there is no duplicate property in two different `configurationsPerComponentType` to avoid redundancy.

```
{
  "type" : "error",
  "configurationsPerComponentType" :
  [
    {
      "componentTypes" : ["SERVLET", "JAXRS", "SOA"],
      "excludeReturnCodes" : {},
      "excludeReturnCodesRegex" : {},
      "excludeErrors" : {"java.lang.RuntimeException" : ["/hello/
errorConfigTest/throwException .*"]},
      "excludeErrorMessages" : {},
      "includeReturnCodes" : {},
      "includeReturnCodesRegex" : {},
      "includeErrors" : {},
      "includeErrorMessages" : {},
```

```
      "includeFirst" : false
    },
    {
      "componentTypes" : ["SERVLET"],
      "excludeReturnCodes" : {"HTTP 500" : ["/hello/errorConfigTest/
throwException (GET)"]},
      "excludeReturnCodesRegex" : {},
      "excludeErrors" : {},
      "excludeErrorMessages" : {},
      "includeReturnCodes" : {},
      "includeReturnCodesRegex" : {},
      "includeErrors" : {},
      "includeErrorMessages" : {},
      "includeFirst" : false
    }
  ]
}
```

Enable Privacy Settings

Enable Privacy Settings to stop APM from storing personal identifiable information.

Storage of certain data can be toggled in these settings like:

- Storing full URLs, not only the domains
- Storing page titles and click names

To enable or disable privacy settings:

1. Log into Oracle Management Cloud and click APM.
2. In the APM Admin page, click Privacy Settings.

Note that Web Application data will not be affected. Follow instructions on the configuration page to configure IP address reporting and URL related privacy settings based on your reporting requirements. It is strongly recommended to avoid collecting any personally identifiable information.

Create Alert Rules

The Oracle Application Performance Monitoring administrator can create alert rules from the Alerts Home Page.

Alerts are created for fixed thresholds, anomalies and early warnings for metrics on Pages, AJAX calls, and Server Requests. To create alert rules:

1. In the Alerts page, click **Alert Rules**.
2. Select APM in the **Service** drop-down to create a rule in Oracle Application Performance Monitoring.
3. Click **Create Alert Rule**.
4. Specify the values for your alert rule.
    a. Specify a **Rule Name**.
    b. Click **Add Description** and add a description for the rule.

c. Click **Add Entities**. Specify details of the entity in the Add Entities window, select an Entity Type and click **Add**.

- Select an entity definition — By Entity Type, By Application, or Individually.
- If you selected Entity Type, choose one or more types of entity.

  If you selected Application, choose one or more application. This will help you create an alert for an entity of a specific application.

  If you selected Individually, select an entity type and choose one or more entities.

  Click **Add All** or **Add Selected**.

The new entity is listed under Entities. You can add another entity, or edit or delete an entity.

d. Click **Add Condition** to specify the trigger for the alert.

e. In the Add Condition window, choose the condition type and specify the parameters.

i. Fixed Metric: Fixed metric conditions will trigger warning and/or critical alerts and notifications based on the threshold values you provide in the condition. They will send the alerts when the system detects that the threshold has been crossed for a particular metric on a particular entity.

- Choose a **Metric** on which to base the condition.
- Choose an **Operator**, specify a **Warning Threshold** and a **Critical Threshold**.
- Specify a time in the **Generate alerts when the metric is outside the specified threshold for __ minutes** field and click **Add**.

ii. Anomaly: Anomaly conditions will trigger alerts and related notifications when the system detects anomalous behavior deviations from the historical baseline.

- Choose a **Metric** on which to base the condition.
- Click **Add**.

iii. Early Warning: Early warning conditions will trigger a warning alert when, based on historical performance, the system predicts the provided threshold will be crossed in the future.

- Choose a **Metric** on which to base the condition.
- Choose an **Operator** and specify a **Warning Threshold**.
- Click **Add**.

The condition is listed in the Alert Conditions list. You can add another condition, or edit or delete a condition.

f. Optionally, provide one or more **Email** addresses to which email notifications will be sent when the alert gets created.

g. Optionally, provide one or more **User Names** to which mobile notifications will be sent when the alert gets created.

5. Click **Save**.

Your new alert rule is displayed in the Alert Rules page. If an event that matches the listed specifications occurs, an alert is displayed in the Alerts page. You can click the alert message or the entity in the alert to view details.

Monitor a Web Application through Servlet Monitoring

Oracle Application Performance Monitoring is capable of monitoring Java web applications by instrumenting the application's Filters and Servlets, through Servlet Monitoring.

When a request is made to a web application / Servlet, a Server Request will be created with type **SERVLET**. The name of this Server Request will be the request URL path, without the schema, port, host, and query parameters.

For example, if a user calls `http://localhost:7654/contextRoot/test.jspx?param1=value1&param2=value2`, the default Server Request name will be `/contextRoot/test.jspx`.

Topics:

Frameworks

Servlet Monitoring detects some popular UI frameworks automatically and appends the name of the framework to the Server Request type. For example, if the framework detected is JSF, the Server Request type will be `SERVLET_JSF`. Here is the list of frameworks that are detected:

- STRUTS1
- STRUTS2
- JSF
- SPRING_MVC
- JERSEY
- APACHE_CXF
- RESTEASY
- APACHE_WINK
- RESTLET
- LIFERAY
- Oracle ADF-UI

  Note: For the framework Oracle ADF-UI, the Server Request type will be `SERVLET_ADF_UI`.

Special Cases

There are two special Servlet related Server Requests that are used to prevent an excessive number of unique Server Requests.

- ***Not-Found:*** Groups together Server Requests that resulted in a HTTP 404 (Not Found) response status.
- ***Invalid-URL:*** Groups together Server Requests that resulted in a HTTP 500 response status due to using an invalid URL. For example, you will see this Server Request if the following Servlet frameworks are used with invalid URLs:
  - JSF (JavaServer Faces) 2.x
  - Struts 1.x

Configuring Servlet Monitoring

You can configure Servlet Monitoring to prevent monitoring of certain pages/resources in your application.

Without a configuration, Servlet Monitoring would monitor all requests to the web application. In many cases, however, you may not want to monitor all requests. For example, you might not want to monitor requests for static resources such as images and HTML files. For this reason, there is a default servlet configuration that is configured to not monitor requests with the following file extensions: `bmp`, `css`, `png`, `swz`, `jar`, `htm`, `html`, `dtd`, `mpeg`, `jpg`, `dat`, `mpg`, `mid`, `properties`, `js`, `ico`, `class`, `tif`, `gif`, `jpeg`, `swf`, `cur`, and `woff`. This selective monitoring is specified by way of a snippet of JSON configuration code that appears in the default servlet configuration:

```
{
    "type" : "servlet",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar",
".htm", ".html", ".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid",
".properties", ".js", ".ico", ".class", ".tif", ".gif", ".jpeg",
".swf", ".cur", ".woff" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
bea_wls_deployment_internal", "/empbs" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default",
        "patternReplacements" : [
            {"patternString" : "[0-9][0-9][0-9][0-9][:-][01]?[0-9][:-]
[0-9]?[0-9](T)?", "replaceString" : "*$1"},
            {"patternString" : "[012]?[0-9]:[0-9][0-9]:[0-9][0-9](Z)?",
"replaceString" : "*$1"},
            {"patternString" : "/_/.*", "replaceString" : "/*"},
            {"patternString" : "/[0-9a-zA-Z_\\.]{14,}[=|-]{1,2}(/|$)",
"replaceString" : "/*$1"},
```

```
            {"patternString" : "([a-fA-F\\._\\:-]*[0-9]+){2,}[a-fA-F_\
\:-]*([/\\.])?", "replaceString" : "*$2"}
        ]
    }
```

You can edit this default JSON configuration code in order to control the types of requests that are monitored. You can also add JSON values to control how your application's Server Request names are abbreviated for display in the Oracle Application Performance Monitoring UI (Server Request names are derived from request URLs).

The servlet configuration can be found in two places:

- `<DOMAIN_HOME>/apmagent/config/Servlet.json` - If this file is modified, all agents on this domain will be updated to use the new configuration.
- `<DOMAIN_HOME>/apmagent/config/<server-name>/Servlet.json` - If this file is modified, only the agent on that specific server will be updated to use the new configuration.

After the configuration is changed, you need not restart the application server, as the agent will automatically detect the change, and update within a minute.

- To see a list of JSON properties that can be used in your configuration code, see Servlet Configuration Options.
- To see a list of examples of configuration, see Examples of Servlet Configuration.

Browser Agent Configuration

Browser Agent Configuration uses the same format as the Servlet Configuration, but targets **Pages** instead of Server Requests. When a Page is excluded, Browser Injection will not be performed on it, which will cause the Page and its AJAX Calls to not be monitored. This configuration uses the following options:

- Extension exclusion (extensionGroup and extensionGroups)
- excludedContextRoots
- excludedServletClasses
- excludedServletPackages
- Pattern Matching

The *type* option determines if the configuration targets Servlet Monitoring or the Browser Agent. Servlet uses `servlet` and Browser Agent uses `browseragent`. For example, here is the default `BrowserAgent.json`:

```
{
    "type" : "browseragent",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar", ".htm",
".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid", ".properties", ".js",
".ico", ".class", ".tif", ".gif", ".jpeg", ".swf", ".cur", ".woff" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
bea_wls_deployment_internal", "/empbs" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
```

```
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default"
    }
}
```

Note the *type* at the top. In addition, the default browser agent configuration does not exclude `.html` by default, since `.html` is necessary to monitor some web application frameworks properly (such as Jet).

The Browser Agent configuration can be found in two places:

- `<DOMAIN_HOME>/apmagent/config/BrowserAgent.json` - If this file is modified, all agents on this domain will be updated to use the new configuration.
- `<DOMAIN_HOME>/apmagent/config/<server-name>/BrowserAgent.json` - If this file is modified, only the agent on that specific server will be updated to use the new configuration.

If the `BrowserAgent.json` is removed and the application server is restarted, it will use the `Servlet.json` configuration instead.

Servlet Configuration Options

You can specify various servlet configuration options to change how your web application / servlet is being monitored by Oracle Application Performance Monitoring.

Topics:

ServletConfiguration

The following is the outermost object in the `.json` file, and represents the `ServletConfiguration` class:

```
{
    "type": "servlet",
    "extensionGroups": {
        "default" : [ ".htm", ".html", ".dtd", ".mpeg", ".jpg",
".dat" ],
        "test" : [".js", "htm", "html" ]
    },
```

```
    "excludedContextRoots": [ "/console", "/em"],
    "excludedFrameworks": [ "JERSEY", "STRUTS1" ],
    "excludedServletClasses": [ "com.siebel.analytics.web.SAWBridge",
"oracle.j2ee.ws.server.provider.ProviderServlet"],
    "excludedServletPackages": [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig": {},
    "frameworkConfigs": {
        "ADF_UI" : {},
        "STRUTS2" : {}
    },
    "deploymentConfigs": {
        "/queryRunner" : {},
        "/simpleServlet" : {}
    },
    "urlConfigs": [{}, {}]
}
```

Here is the list of classes in the above code, and their description.

Class	Description
`type` (String)	Indicates the type of JSON configuration. The servlet configuration uses `servlet`. The Browser Agent configuration, which uses the same JSON format uses `browseragent`.
`extensionGroups` (Map)	**Key** is the group name, **value** is the array of file extensions (For example, .jpg, .png, .gif). Extensions that do not start with a "." will have it added in the agent run-time. The group name is used in the `ScopeConfiguration` object.
`excludedContextRoots` (Array)	List of context-roots that should not be monitored. Context-roots that do not start with a "/" will have it added in the agent run-time.
`excludedFrameworks` (Array)	List of frameworks that should not be monitored. You can exclude any of these frameworks: STRUTS1, STRUTS2, JSF, SPRING_MVC, JERSEY, APACHE_CXF, RESTEASY, APACHE_WINK, RESTLET, LIFERAY, ADF_UI. Through the `apmUi` you can specify which Server Requests are related to which frameworks based on the type (`SERVLET_FRAMEWORK_NAME`).
`excludedServletClasses` (Array)	List of full servlet class names that should not be monitored.
`excludedServletPackages` (Array)	List of servlet Java packages that should not be monitored.
`defaultConfig` (Object)	`ScopeConfiguration` object. This will be used if a request does not use any of the specified `frameworkConfigs`, `deploymentConfigs`, or `urlConfigs`.
`frameworkConfigs` (Map)	Key is framework name. Value is `ScopeConfiguration` object. Currently supported framework names: STRUTS1, STRUTS2, JSF, SPRING_MVC, GWT, JERSEY, APACHE_CXF, RESTEASY, APACHE_WINK, RESTLET, LIFERAY, ADF_UI. This can be used to create a configuration that targets all requests that are from a particular framework. `deploymentConfigs` and `urlConfigs` take precedence over this.

Class	Description
`deploymentConfigs` (Map)	Key is context-root. Value is `ScopeConfiguration` object. This can be used to create a configuration that targets all requests that are for a particular deployment/application. `urlConfigs` takes precedence over this.
`urlConfigs` (Array)	List of `UrlConfiguration` objects. This can be used to create a configuration that targets all requests with URLs that fit a particular `regex` pattern.

UrlConfiguration

The `UrlConfiguration` is used to create `ScopeConfigurations` that are based on a `regex` pattern, instead of a framework name or context-root.

> **Note:**
>
> This check is run on every request. Use this option only if the context root is not sufficient.

```
{
    "patternString" : ".*/test/.*",
    "config" : {}
}
```

Class	Description
`patternString` (String)	A `regex` pattern. This is used to determine if the specified `config` should be used.
`config` (Object)	`ScopeConfiguration` object.

ScopeConfiguration

`ScopeConfiguration` is the configuration that targets a specific scope (framework, deployment, URL).

```
{
    "extensionGroup" : "default",
    "patternMatching" : [{}, {}],
    "patternMappings" : [{}, {}],
    "patternReplacements" : [{}, {}],
    "frameworkOptions" : {},
    "parameterGroups" : [ [ "param5" ], [ "param1", "param4",
"param2" ], [ "param3", "param1", "param2" ], ["param6"] ],
    "headerGroups" : [ [ "header5" ], [ "header1", "header4",
"header2" ], [ "header3", "header1", "header2" ], ["header6"] ]
}
```

Class	Description
`extensionGroup` (String)	Specifies which extension group this particular scope uses. All extensions in the specified extension group will be excluded.
`patternMatching` (Array)	List of `PatternMatcher` objects. It works on a *match first* system. The first pattern that matches the URL determines if it is excluded or not.
`patternMappings` (Array)	List of `PatternMapping` objects.
`patternReplacements` (Array)	List of `PatternReplacement` objects.
`frameworkOptions` (Object)	`FrameworkOptions` object.
`parameterGroups` (Array)	Array of arrays. Each group contains a list of parameters (GET or POST). Servlet Monitoring will check each group in the order specified. If all of the parameters in the group are present, it will add them all to the Server Request name in a format similar to query strings (`?param1=value1&param2=value2`). If at least one is missing, it will move onto the next group.
`headerGroups` (Array)	Array of arrays. Each group contains a list of HTTP request header names. Servlet Monitoring will check each group in the order specified. If all of the headers in the group are present, it will add them all to the Server Request name in the following format: `[header1=value1] [header2=value2]`

PatternMatcher

This is used to help filter out particular requests by using URL-based `regex` patterns.

```
{
    "patternString" : ".*/patternMatching/discover/this",
    "excluded" : false,
    "includeQueryString": false
}
```

Class	Description
`patternString` (String)	A `regex` pattern.
`excluded` (Boolean)	If true, the URL will be excluded if it matches this pattern. If false, the URL will be monitored if it matches this pattern. Defaults to false if unspecified.
`includeQueryString` (Boolean)	If true, the query string at the end of the URL will be included when applying the pattern. Defaults to false if unspecified.

PatternMapping

This is used to help alter request URLs that match a specified URL-based `regex` pattern. In particular, it helps group REST-like URLs into a single Server Request.

```
{
    "patternString" : ".*/patternMapping/convertThis",
    "formatString" : "/patternMapping/intoThis",
    "formatType" : "BASIC",
```

```
    "includeQueryString" : false
}
```

Class	Description
patternString (String)	A regex pattern.
formatString (String)	Used to create a MessageFormat object. You can specify "{#}" in a format to insert particular values. These values are determined by the formatType setting.
formatType (String)	Determines how the Server Request name is formatted. Options are: BASIC, POSITION, and GROUP.
includeQueryString (Boolean)	If true, the query string at the end of the URL will be included when applying the pattern. Defaults to false if unspecified. For example, in the URL http://localhost:7654/simpleServlet/test/1/2/3?param1=value1&param2=value2, only /simpleServlet/test/1/2/3 will be applied to the regex pattern by default. If includeQueryString is true, it will include ?param1=value1&param2=value2.

PatternReplacement

This is used to help alter request URLs that contain one or more specified regex patterns. All specified PatternReplacements will be applied to the request URL. This is useful for removing common patterns across multiple URL formats, such as dates, IDs, and numbers.

```
{
    "patternString" : "/[0-9]+/",
    "replaceString" : "/num/"
}
```

Class	Description
patternString (String)	A regex pattern.
replaceString (String)	Used to replace each occurrence the regex pattern matches. Note that this utilizes Matcher.replaceAll, so if the regex pattern has a group specified, it can be referenced here using $1, $2, etc.

FrameworkOptions

This is a collection of framework specific options. Currently, options are available only for ADF-UI.

```
{
    "adf": {}
}
```

Class	Description
adf (Object)	AdfOptions object

AdfOptions

ADF-UI specific configuration options. Some ADF-UI applications are setup to run under a single page, which can cause all requests to fall under a single Server Request by default. These options will help you split apart the Server Request.

All the options utilize the `oracle.adf.view.rich.monitoring.UserActivityInfo` parameter. The available options will append the following three properties from the `UserActivityInfo`: `regionViewId`, `componentClientId`, and `eventType`.

If all the three properties are enabled, the format/order will be:

```
/server/request/name/{regionViewId}/{componentClientId}/{eventType}
```

```
{
    "appendRegionViewId": true,
    "removeActivityId": false,
    "componentClientIds" : [ "r1:0:sayHello", "r2:0:sayGoodbye" ],
    "eventTypes" : [ "action", "dialog" ],
    "appendAllComponentClientIds": false,
    "appendAllEventTypes": false
}
```

Class	Description
`appendRegionViewId` (Boolean)	Appends the `regionViewId` value from `UserActivityInfo` to the Server Request name if true.
`removeActivityId` (Boolean)	`regionViewId` typically has the following format: `taskFlowName/activityId`. If `removeActivityId` is true, the `/activityId` will be stripped out. `appendRegionViewId` must also be true.
`componentClientIds` (Array)	List of `componentClientIds` that, if detected, will be appended onto the end of the Server Request name. Specified `componentClientIds` will be stripped of sections that contain only numbers. For example, `r1:0:sayHello` will be reduced to `r1:sayHello`.
`eventTypes` (Array)	List of `eventTypes` that, if detected, will be appended onto the end of the Server Request name.
`appendAllComponentClientIds` (Boolean)	Appends the `componentClientId` value from `UserActivityInfo` to the Server Request name if true. Overrides `componentClientIds`.
`appendAllEventTypes` (Boolean)	Appends the `eventType` value from `UserActivityInfo` to the Server Request name if true. Overrides `eventTypes`.

By default, an ADF-UI application will not provide the `oracle.adf.view.rich.monitoring.UserActivityInfo` parameter. To enable it, add the following `context-param` to the application's `web.xml`:

```
<context-param>
  <description>
    This parameter notifies ADF Faces that the ExecutionContextProvider
    service provider is enabled. When enabled, this will start
    monitoring and aggregating user activity information for the client
    initiated requests. By default, this param is not set or is false.
  </description>
  <param-name>
```

```
    oracle.adf.view.faces.context.ENABLE_ADF_EXECUTION_CONTEXT_PROVIDER
  </param-name>
  <param-value>true</param-value>
</context-param>
```

Examples of Servlet Configuration

Given below are examples of various configuration options.

- Multiple Scopes
- Pattern Matching
- Pattern Mapping
- Pattern Replacement
- Partition By Parameter/Header
- ADF-UI

Multiple Scopes

If multiple scopes are configured, `ScopeConfigurations` are checked in the following order, and work on a *match-first* system: **url > deployment > framework > default**. The following actions occur on the agent:

1. Request is picked up by Servlet Monitoring.
2. Request URL is checked against any specified `urlConfig` pattern strings. If a match is found, that `ScopeConfiguration` will be used. Otherwise, the request moves on.
3. Then, the request moves onto deployments. If the context-root matches any specified under `deploymentConfigs`, that `ScopeConfiguration` will be used. Otherwise, the request moves on.
4. Next, the request moves onto frameworks. If a framework is detected, and it matches any specified under `frameworkConfigs`, that `ScopeConfiguration` will be used. Otherwise, the request moves on.
5. Finally, the request will use the `defaultConfig` if nothing else matched.

Below is an example of multiple scopes:

```
{
    "type" : "servlet",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar",
".htm", ".html", ".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid",
".properties", ".js", ".ico", ".class", ".tif", ".gif", ".jpeg",
".swf", ".cur", ".woff" ],
        "framework" : [ ".json" ],
        "deployment" : [ ".docx" ],
        "url" : [ ".fake" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
bea_wls_deployment_internal", "/empbs" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
```

```
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default"
    },
    "frameworkConfigs": {
        "ADF_UI" : {
            "extensionGroup" : "framework"
        }
    },
    "deploymentConfigs": {
        "/simpleServlet" : {
            "extensionGroup" : "deployment"
        }
    },
    "urlConfigs": [{
        "patternString" : ".*/specialCase/.*",
        "config" : {
            "extensionGroup" : "url"
        }
    }]
}
```

Here are some example URLs and the result of the above configuration:

1. `http://localhost:7654/simpleServlet/test (context-root: /simpleServlet, framework: none)`
   - `/simpleServlet/test` does not fit the `.*/specialCase/.*` pattern, so that is skipped.
   - The `deploymentConfig` settings are specified for `/simpleServlet`. For this request, the *deployment* `extensionGroup` will be used.
   - Request is not excluded because it doesn't end with `.docx`.
2. `http://localhost:7654/a/faces/specialCase/test.fake (context-root: /a, framework: ADF-UI)`
   - `/a/b/c/specialCase/test.fake` fits the `.*/specialCase/.*` pattern. For this request, the url `extensionGroup` will be used.
   - Even though the framework of the request is ADF-UI (which has a `frameworkConfig`), `urlConfig` was matched first, so it takes priority.
   - Request ends up getting excluded because `.fake` is in the *url* `extensionGroup`.
3. `http://localhost:7654/b/faces/another/case.fake (context-root: /b, framework: ADF-UI)`
   - `/b/faces/another/case.fake` doesn't match the `urlConfig` pattern.
   - It also doesn't have the context-root `/simpleServlet`.
   - It does, however, use the ADF-UI framework. For this request, the *framework* `extensionGroup` will be used.

- Request is not excluded because it doesn't end with `.json`.

4. `http://localhost:1234/simpleServlet/test (context-root: /, framework: none)`
   - `/simpleServlet/test` doesn't match the urlConfig's pattern.
   - This is an important step - While the URL does start with the `/simpleServlet`, it is not the context-root. In this scenario, the deployment is using a blank context-root ("/"). As a result, it doesn't match the `deploymentConfig`.
   - It also doesn't match the `frameworkConfig`.
   - Since it did not match other configurations, the `defaultConfig` is used. For this request, the "default" `extensionGroup` will be used.
   - Request is not excluded because it doesn't end with any of the extensions listed in the `default` group.

Pattern Matching

Pattern matching helps filter out requests that you do not want to monitor.

```
{
    "type" : "servlet",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar",
".htm", ".html", ".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid",
".properties", ".js", ".ico", ".class", ".tif", ".gif", ".jpeg",
".swf", ".cur" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
bea_wls_deployment_internal" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default",
        "patternMatching" : [ {
            "patternString" : ".*/patternMatching/discover/this",
            "excluded" : false
        }, {
            "patternString" : ".*/patternMatching/and/discover/this",
            "excluded" : false
        }, {
            "patternString" : ".*",
            "excluded" : true
        } ]
    }
}
```

Servlet Monitoring will step through each pattern in the order specified. It uses *match-first* logic. For example, if the request URL is `http://localhost:7654/`

`simpleServlet/patternMatching/and/discover/this` and we use the configuration above:

- `/simpleServlet/patternMatching/and/discover/this` would be applied to the first pattern, and fail.
- It would then be applied to the next pattern. This time it matches. This particular pattern has *excluded* set to *false* (that is, *included*). This means that the request will be monitored.
- Despite the third pattern excluding everything, it was already matched against the second pattern, so the process is complete.

With the way the three patterns are setup, only URLs that end with `/patternMatching/discover/this` and `/patternMatching/and/discover/this` would be monitored. Every other request would be excluded by the third pattern.

> **Note:**
>
> If a pattern fails to match any of the specified patterns, it will be considered `included`.

Pattern Mapping

Pattern mapping helps manipulate request URLs into more manageable Server Request names in Oracle Application Performance Monitoring. This is handy for handling things such as REST URLs, which commonly have parameters in the URL itself.

```
{
    "type" : "servlet",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar",
".htm", ".html", ".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid",
".properties", ".js", ".ico", ".class", ".tif", ".gif", ".jpeg",
".swf", ".cur" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
bea_wls_deployment_internal" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default",
        "patternMappings" : [ {
            "patternString" : ".*/patternMapping/convertThis",
            "formatString" : "/patternMapping/intoThis",
            "formatType" : "BASIC"
        }, {
            "patternString" : ".*/patternMapping/this/is/a/path",
            "formatString" : "/{2}/{5}/{6}/{3}/{4}",
            "formatType" : "POSITION"
```

```
        }, {
            "patternString" : ".*/patternMapping/regexGroup/(.*)/blank/
(.*)",
            "formatString" : "/patternMapping/{2}/{1}",
            "formatType" : "GROUP"
        } ]
    }
}
```

For example, let's say there is a REST service that accepts the following URL format: `http://localhost:7654/calculator/add/{value1}/{value2}`. By default all variations of `value1` and `value2` will be its own Server Request. A pattern mapping can be created to merge all of these into a single `/calculator/add` Server Request.

In the above example, there are three pattern mapping examples, which use different `formatType` values: BASIC, POSITION, and GROUP.

- Example 1 (BASIC) is simple: any URLs that match the pattern will always use `/patternMapping/intoThis` as the Server Request name.
- Example 2 (POSITION) is a bit more complicated. In the URL path `/cr/patternMapping/this/is/a/path` this is the URL position for each part of the URL:
  1. cr
  2. patternMapping
  3. this
  4. is
  5. a
  6. path

  When this is applied to the `formatString`, we get `/patternMapping/a/path/this/is`.
- Example 3 (GROUP) uses Regex groups. In the URL path `/cr/patternMapping/regexGroup/example/blank/test`, `group 1` will be `example`, and `group 2` will be `test`. When applied to the format, this becomes `/patternMapping/test/example`.

Pattern Replacement

Pattern replacement helps manipulate request URLs into more manageable Server Request names in Oracle Application Performance Monitoring. This is handy for handling common patterns that may appear in URLs such as IDs (numbers, Hex, Base64), Dates, Times, etc.

```
 {
    "type" : "servlet",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar",
".htm", ".html", ".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid",
".properties", ".js", ".ico", ".class", ".tif", ".gif", ".jpeg",
".swf", ".cur", ".woff" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
```

```
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
bea_wls_deployment_internal", "/empbs" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default",
        "patternReplacements" : [
            {"patternString" : "/[0-9]+/", "replaceString" : "/num/"},
            {"patternString" : "test([A-Z])", "replaceString" : "t$1"}
        ]
    }
}
```

For example, let's use the following URL: `http://localhost:7654/testA/1234/testB/4321/testC`

The above configuration specified two `patternReplacements`, which will be applied to the URL path in the order specified.

Start: `/testA/1234/testB/4321/testC` **First replacement:** `/testA/num/testB/num/testC` **Second replacement:** `/tA/num/tB/num/tC`

The Server Request name will end up being `/tA/num/tB/num/tC` for that URL.

Partition By Parameter/Header

Here are a few example request URLs, and how the configuration alters them:

```
{
    "type" : "servlet",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar",
".htm", ".html", ".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid",
".properties", ".js", ".ico", ".class", ".tif", ".gif", ".jpeg",
".swf", ".cur" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
bea_wls_deployment_internal" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default",
        "parameterGroups" : [ [ "param5" ], [ "param1", "param4",
"param2" ], [ "param3", "param1", "param2" ], ["param6"] ],
        "headerGroups" : [ [ "header5" ], [ "header1", "header4",
"header2" ], [ "header3", "header1", "header2" ], ["header6"] ]
    }
}
```

1. `http://localhost:7654/simpleServlet/testingParams?param1=value1&param2=value2&param3=value3` (no headers)
   - The first parameter group is checked. `param5` is not set, so the group is skipped.
   - The second parameter group is checked. `param1` is set, but `param4` is not, so the group is skipped (doesn't reach the check for `param2`).
   - The third parameter group is checked. `param3`, `param1`, and `param2` are all set, so this group will be used.
   - The values of all three parameters are acquired and placed in the final Server Request name (in the order specified): `/simpleServlet/testingParams?param3=value3&param1=value1&param2=value2`
2. `http://localhost:7654/simpleServlet/testingParams?param5=value5 (Headers: header6 = headerValue6)`
   - The first parameter group is checked. `param5` is set, so this group will be used.
   - The first three header groups are skipped because none of them are set.
   - The final group is checked. `header6` is set, so this group will be used.
   - The values of both the parameter and header are acquired and place in the final Server Request name: `/simpleServlet/testingParams?param5=value5 [header6=headerValue6]`
3. `http://localhost:7654/simpleServlet/testingParams (Headers: header1=headerValue1, header2=headerValue2, header3=headerValue3, header4=headerValue4)`
   - None of the parameter groups are used.
   - The first header group is checked. `header5` is not set, so the group is skipped.
   - The second header group is checked. All three headers (1, 4, and 2) are present, so this group will be used.
   - Despite the third header group having three present headers (3, and 1, and 2), we already found a match, so it is not checked.
   - The values of all three headers are acquired and placed in the final Server Request name (in the order specified): `/simpleSerlvet/testingParams [header1=headerValue1][header4=headerValue4][header2=headerValue2]`

ADF-UI

Example:

```
{
    "type" : "servlet",
    "extensionGroups" : {
        "default" : [ ".bmp", ".css", ".png", ".swz", ".jar",
".htm", ".html", ".dtd", ".mpeg", ".jpg", ".dat", ".mpg", ".mid",
".properties", ".js", ".ico", ".class", ".tif", ".gif", ".jpeg",
".swf", ".cur" ]
    },
    "excludedContextRoots" : [ "/bea_wls_management_internal2",
"/bea_wls_cluster_internal", "/bea_wls_internal", "/soa-infra", "/
```

```
bea_wls_deployment_internal" ],
    "excludedServletClasses" : [ "com.siebel.analytics.web.SAWBridge",
"oracle.j2ee.ws.server.provider.ProviderServlet",
"oracle.webcenter.content.http.GetHandlerServlet" ],
    "excludedServletPackages" : [ "weblogic.wsee.jaxws.",
"weblogic.wsee.server.servlet." ],
    "defaultConfig" : {
        "extensionGroup" : "default",
        "frameworkOptions" : {
            "adf": {
                "appendRegionViewId": true,
                "removeActivityId": true,
                "appendAllComponentClientIds": true,
                "appendAllEventTypes": true
            }
        }
    }
}
```

Here is an example of the `oracle.adf.view.rich.monitoring.UserActivityInfo` parameter:

```
<m xmlns="http://oracle.com/richClient/comm">
  <k v="previous">
    <m>
      <k v="_contextId">

<s>bf24897846148f26:7253b46:139c113330b:-8000-00000000000005a6</s>
      </k>
      <k v="_clientStartTime">
        <s>1347578879624</s>
      </k>
      <k v="_clientEndTime">
        <s>1347578898966</s>
      </k>
    </m>
  </k>
  <k v="secondary">
    <m>
      <k v="_contextId">

<s>bf24897846148f26:7253b46:139c113330b:-8000-00000000000005a6:_adfStrea
ming</s>
      </k>
      <k v="_parentContextId">

<s>bf24897846148f26:7253b46:139c113330b:-8000-00000000000005a6</s>
      </k>
    </m>
  </k>
  <k v="primary">
    <m>
      <k v="_clientStartTime">
        <s>1347578919325</s>
```

```
      </k>
      <k v="_eventInfo">
        <m>
          <k v="eventType">
            <s>action</s>
          </k>
          <k v="componentClientId">
            <s>ptemp:r1:1:tbb1:1:_afrButtonStopNavItem</s>
          </k>
          <k v="componentType">
            <s>oracle.adf.RichCommandTrainStop</s>
          </k>
          <k v="componentDisplayName"/>
          <k v="regionViewId">
        <a n="1">
              <s>/customer-registration-task-flow/defineAddresses</s>
            </a>
          </k>
          <k v="regionViewName">
            <a n="1">
              <s>ptemp:r1</s>
            </a>
          </k>
        </m>
      </k>
      <k v="_pprTargets">
        <s>ptemp:r1</s>
      </k>
    </m>
  </k>
</m>
```

Servlet Monitoring supports both the new and old formats of `UserActivityInfo` (newer format is more optimized, replacing longer strings with shorter versions. For example: `regionViewId --> rvd, componentClientId --> cld`, etc.) If the URL was `http://localhost:7654/adfTest/faces/test.jspx` with the above `UserActivityInfo` and servlet configuration, the following would occur:

- `appendRegionViewId` is true, so the `regionViewId` value is used. In this case, it is `/customer-registration-task-flow/defineAddresses`.
- `removeActivityId` is also true. The `activity id` in this case is the `/defineAddresses` part of the `regionViewId`. The result is now `/customer-registration-task-flow`.
- Next, `appendAllComponentClientIds` is true, so the `componentClientId` value is used. In this case, it is `ptemp:r1:1:tbb1:1:_afrButtonStopNavItem`. The `:{number}:` sections of the `componentClientId` are removed, resulting in `ptemp:r1:tbb1:_afrButtonStopNavItem`.
- Finally, `appendAllEventTypes` is true, so the `eventType` value is used. In this case, it is `action`.
- The three values are now appended onto the end of the Server Request name (separated by "/"). The final Server Request name

ends up being `/adfTest/faces/test.jspx/customer-registration-task- flow/ptemp:r1:tbb1:_afrButtonStopNavItem/action.`

Set Up Custom Instrumentation

Use Custom Instrumentation to add monitoring capabilities to technologies not supported by Oracle Application Performance Monitoring.

Why Use Custom Instrumentation?

- Technology used in application is not supported by Oracle Application Performance Monitoring.
- To obtain finer granularity of application performance metric breakdown.

Custom Instrumentation Guidelines

Custom Instrumentation is enabled by having the file **custom-pointcuts.properties** in Oracle Application Performance Monitoring Agent root config directory or appserver specific config directory.

For example, if the agent is provisioned to **`<weblogic_domain_home>`**, then **`custom-pointcuts.properties`** is automatically read from the location below:

```
<weblogic_domain_home>/apmagent/config/<appserver_name>/custom-pointcuts.properties
```

If the file name of custom pointcuts properties is other than **custom-pointcuts.properties** or it is placed in any directory other than appserver specific config directories, then add the following property to the JVM startup argument with the path to pointcuts file to enable custom instrumentation:

```
oracle.apmaas.agent.custom.pointcuts=<path_to_custom_pointcuts_file>
```

If Java system property is added to java executable argument, it is prefixed with -D. For example:

```
-Doracle.apmaas.agent.custom.pointcuts=<path_to_custom_pointcuts_file>
```

> **Note:**
>
> **Log info on custom pointcut:**
> When agent found and read custom pointcuts properties file, a message gets printed to standard out:
>
> "APMCS agent - INFO: Reading custom pointcut file [file=<path/to/custom/pointcuts/properties/file>]"

Enable Method Invocation

To enable Method Invocation in Custom Instrumentation, follow these steps:

1. Open the `AgentStartup.properties` file:

   ```
   <weblogic_domain_home>/apmagent/config/<appserver_name>/
   AgentStartup.properties
   ```

2. Enable Method Invocation by editing the following line:

   ```
   oracle.apmaas.agent.probe.custom.enableMethodInvocation=true
   ```

Use the Thread Profiler

Use the thread profiler to identify key classes and methods you want to include in Custom Instrumentation.

Oracle Application Performance Monitoring includes a Thread Profiler to retrieve thread dump of JVM the APM Agent is running in. The steps below go over how to get the stack trace of an appserver monitored by Oracle Application Performance Monitoring.

If the classes and methods have already been identified, proceed to enable Custom Instrumentation. The full class and methods names will be needed.

1. In the Oracle Management Cloud menu, select **Diagnostic Snapshots**, then click **Thread Profiles**.
2. Click **Start Thread Profiler**.
3. Fill in the **Name** field and select and appserver for thread profiling. Click **Start Profiler**.

   The profiler will start in the pending state. Wait for the specified duration and refresh the page.
4. Click the profile name to show the **Thread Profile Summary**.

   The Thread Profile Summary includes the class and methods found from the profiling. The full class and method names will be needed to enable Custom Instrumentation.

Enable Custom Instrumentation

Enable Custom Instrumentation after the classes and methods that need to be monitored are identified.

1. Open a new terminal window in your host monitored by APM agent.
2. Navigate to the APM configuration files:

   ```
   $ cd /<agent_installation>/apmagent/config/<appserver_name>
   ```

3. Create a new file called: **custom-pointcuts.properties**
4. Add the following properties for each of the classes and methods to the file:
   - **`pointcut-<identity>.class=<class name>`**
   - **`pointcut-<identity>.method=<method name>`**

Where: <identity> is an arbitrary unique name (same name for all properties of the same pointcut) to identify a pointcut.

<class name> and <method name> are the full qualified class name and method name obtained from thread profiling.

Repeat step 4 for each of the classes and methods to be monitored.

5. Save the file. Restart the appserver to enable custom instrumentation.

Example 4-1 custom-pointcuts.properties file

```
pointcut-scheduler.class=myapp.scheduler.MySchedule
pointcut-scheduler.method=startProcess
```

This example file adds a scheduler identity for class `myapp` and method `startProcess`.

Custom Instrumentation Reference

Syntax

Property Name	Required	Description	Example
pointcut-<identity>.class	YES	Full qualified class name to be instrumented. Only the specified class is instrumented. Sub class of the specified class can be done with name modifier (See Class Modifier). Also class name support willcard modifier (See Class Modifier).	`myapp.HelloWorld` `myapp.HelloAll`
pointcut-<identity>.method	YES	Name of method to be instrumented. Method name support modifier (See Method Modifier).	`sayHello` `sayHi`
pointcut-<identity>.paramTypes	NO	Comma separated full qualified class name of parameters of the method. If this property is not specified, all methods having the same specified method name of the specified class are instrumented. paramTypes property supports arbitrarily trailing classes (See ParaTypes Modifier)	`java.lang.String` `javax.servlet.Servlet`

Property Name	Required	Description	Example
pointcut-<identity>.operationName	NO	Name of operation. By default, instrumented server request name or operation name is generated by class name and method name. The default can be overridden by this property with a custom name.	`Hello World`

Section Modifier

Type	Modifier	Description	Example
wildcard	*	Character to match any character sequence of class name. If class name property is the wildcard character alone, Custom Instrumentation will instrument all classes which can cause significant performance impact.	`pointcut-hello.class = myapp.*` `pointcut-hello.class = myapp.Hello*`
subclass	+	Prefix class name with + to include subclasses of the specified class.	`pointcut-hello.class = +myapp.HelloInterface`

Method Modifier

Type	Modifier	Description	Examples
wildcard	*	Character to match any character sequence of method name. If method name property is the wildcard character alone, Custom Instrumentation will instrument all methods of specified class.	`pointcut-hello.method = *` `pointcut-hello.method = say*`

Parameter Type Modifier

Type	Modifier	Description	Examples

any trailing classes	...	To represent any number of trailing param classes.	`pointcut-hello.paramTypes = ...`
			`pointcut-hello.paramTypes = java.lang.String, ...`
			`pointcut-hello.paramTypes = java.lang.String, int, ...`

Operation Name Construction

Variable	Description
{class}	Simple name of Instrumented class.
{classFQN}	Full qualified name of instrumented class.
{method}	Instrumented method name.
{paramTypes}	Comma separated mehod parameters in simple class names.
{paramTypesFQN}	Comma separated mehod parameters in full qualified class names.

Example 4-2 Operation Name Construction

```
pointcut-hello.class = myapp.HelloWorld
pointcut-hello.method = sayHello

Here are some examples of operation name constructions:
pointcut-hello.operationName = Say Hello
pointcut-hello.operationName = Say Hello with Name
pointcut-hello.operationName = {class} {method} to everyone
pointcut-hello.operationName = {classFQN}.{method}
pointcut-hello.operationName = {classFQN}.{method}({paramTypes})

Sample:
pointcut-hello.class = myapp.HelloWorld
pointcut-hello.method = sayHello

pointcut-hi.class = myapp.HelloWorld
pointcut-hi.method = sayHi*
pointcut-hi.operationName = Say Hi

pointcut-hey.class = myapp.HeyWorld
pointcut-hey.method = sayHello
pointcut-hey.paramTypes = java.lang.String
pointcut-hey.operationName = Say Hello with Name

pointcut-helloAll.class = myapp.HelloWorld
```

```
pointcut-helloAll.method = sayHello
pointcut-helloAll.paramTypes = java.lang.String[]
pointcut-helloAll.operationName = {classFQN}.{method} to All
```

Avoid invalid custom pointcut configuration:

If the `custom-pointcuts.properties` file is not constructed as per the syntax given above or if the pointcut entries are not configured properly, then custom instrumentation will be ignored and there may not be any message printed to standard output, or logged in any log file.

Common invalid syntax in custom pointcuts properties:

- Mismatch unique pointcut name/id of an entry set
- Reuse of pointcut name/id between multiple entry set
- Typo in class and method names

Example 4-3 Invalid custom.pointcuts.properties files

```
pointcut-hello.class=myapp.HelloWorld

pointcut-hello1.method=sayHello  <== mismatch unique pointcut name


pointcut-hello.class=myapp.HelloWorld   <== reuse of 'hello' unique
name as previous entry set.

pointcut-hello.method=sayHelloToAll  <== reuse of 'hello' unique name
as previous entry set.


pointcut-helloyou.class = myapp.HelloWor   <==  typo in class name

pointcut-helloyou.method = sayHelloYou
```

A

Technologies Supported by Oracle Application Performance Monitoring

Oracle Application Performance Monitoring can monitor applications that run on Java, .Net, Node.js and Ruby.

For the latest and complete support matrix, see Support Note ID 2092363.1.

B

Supported Selenium Commands in Synthetic Tests

Here's a list of supported Selenium commands that you can use for recording scripted actions for your synthetic tests.

Supported Selenium Commands:

- `open`
- `click`
- `clickAt`
- `doubleClick`
- `doubleClickAt`
- `sendKeys`
- `type`
- `setSpeed`
- `storeText`
- `storeTitle`
- `runScript`
- `submit`
- `store`
- `echo`
- `mouseOver`
- `mouseOut`
- `mouseDownAt`
- `mouseMoveAt`
- `mouseUpAt`
- `selectFrame`
- `select`
- `verifyText`
- `verifyTitle`
- `verifyNotText`
- `assertText`
- `assertNotText`
- `assertTitle`

- `setWindowSize`

For description of the above commands, refer to Selenium documentation.