USING APPLICARE › USER EXPERIENCE MONITORING

End User Experience Monitoring (EUM)/User Experience Monitoring (UEM) Analyzer

Applicare 10.0 · Updated August 28, 2024 · **Pending technical review**

The End User Experience Analyzer delivers detailed performance insights into web applications as experienced by actual end users. Applicare agent automatically injects Java script code to JSP pages and that reports back the total time it took to load the page from the time a user clicked the link, network time, server processing and browser rendering times, browser type and user location.

Configuration

By default, Applicare scripts for collecting EUM data are served by the Applicare Controller. However, they can also be served by any Applicare Collector instance. Following JVM arguments need to be set in case a collector is serving these scripts and persisting returned data

Example URLs below assumes that '192.168.1.10' is the IP address of the Applicare Collector and '8080' is the port it is listens on. Please update the IP address and Port as needed adding these arguments and adding them to agent start.

VM Argument	Example & Description
arceum.js	-Darceum.js=`https://192.168.1.10:8080/applicare/eum/arc-eum.js`
	Utilize this parameter to specify from where the Applicare should load arc-eum.js This should be URL to applicare_agent application.
arceumprocess.js	-Darceumprocess.js=`https://192.168.1.10:8080/applicare/eum/arc-eum-process.js`
	Utilize this parameter to specify from where the Applicare should load arc-eum-procss.js This should be URL to applicare_agent application.
eum.beacon	-Deum.beacon=`https://192.168.1.10:8080/applicare/eum/beacon`
	Utilize this parameter to specify on which URL Applicare is listening for UEM data. This should be URL to applicare_agent application.
arc-nav.js	-Darc-nav.js=https://`192.168.1.10:8080/applicare`/eum/arc-nav.js
	If your web application is using servlets and you want to monitor them under UEM then you can utilize this parameter. By default Applicare does.
uem.servlets	-Duem.servlets=true
	If your web application is using servlets and you want to monitor them under UEM then you can utilize this parameter. By default Applicare does

The most crucial step is ensuring that these scripts and beacon URLs are accessible to end users. You might need to adjust firewall rules to permit external access to these URLs.

By setting the parameters mentioned, Applicare will inject JavaScript like the following into pages generated by your application's JSP/servlet when accessed by end users.

```xml
<script src="https://192.168.1.10:8080/applicare_agent/eum/arc-eum.js"
type="text/javascript"></script>

<script type="text/javascript">

BOOMR.addVar({ "server": "AdminServer","t_backend": "4","b_name":
BrowserDetect.browser,"b_ver": BrowserDetect.version,"b_os": BrowserDetect.OS });

BOOMR.init({user_ip: "192.168.1.16",beacon_url:
"https://192.168.1.10:8080/applicare_agent/eum/beacon"});

</script>
```

Next, enable the End User Experience Profiling aspect on your servers to automatically inject JavaScript into JSP pages. Follow these steps:

- Navigate to **Configuration > Configure Dynamic Instrumentation** page in Applicare console.
- On IntelliTrace Deployment tab, select Advanced Mode and click Configure button next to it.
- Move End User Experience Profiling aspect from left side list to Deployed Aspect list by selecting the aspect and clicking Add button.
- Click Apply button and make sure Agent is enabled for servers in IntelliTrace deployment tab.

Programmatic JavaScript Injection

If auto-instrumentation is not available or feasible, you can still implement end user monitoring by manually injecting the necessary scripts into your pages.

Applicare provides an API that allows you to insert the required JavaScript just before the "</body>" tag of your pages.

For instance, you can adjust your application's template to include the following for manual JavaScript injection:

```
... existing template code ...<%= com.arcturus.aop.shared.RUMManager.injectJS() %></body>
```

Additionally, ensure that "<Applicare Installation Dir>/applicare-aspects.jar" is included in your class path during development.

Manual JavaScript Injection for Static Pages

For end user experience monitoring of static pages, Applicare EUM scripts can be manually injected.

URL Exclusion & Inclusion

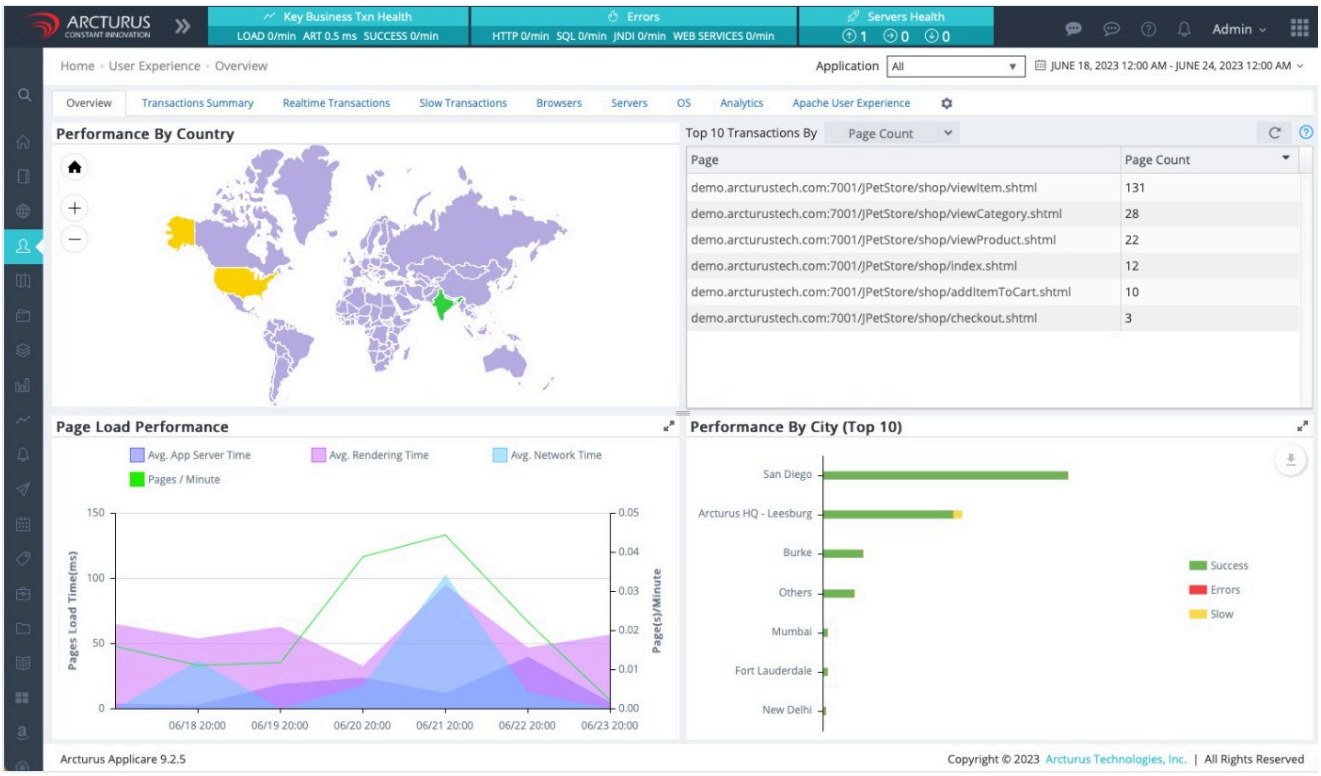
To manage URL monitoring in UEM: Add a URL from the HTTP Analyzer to the UEMURLExclusions.txt file to exclude it from monitoring.

Add a URL from the HTTP Analyzer to the UEMURLInclusions.txt file to include it in monitoring.

Analyzer

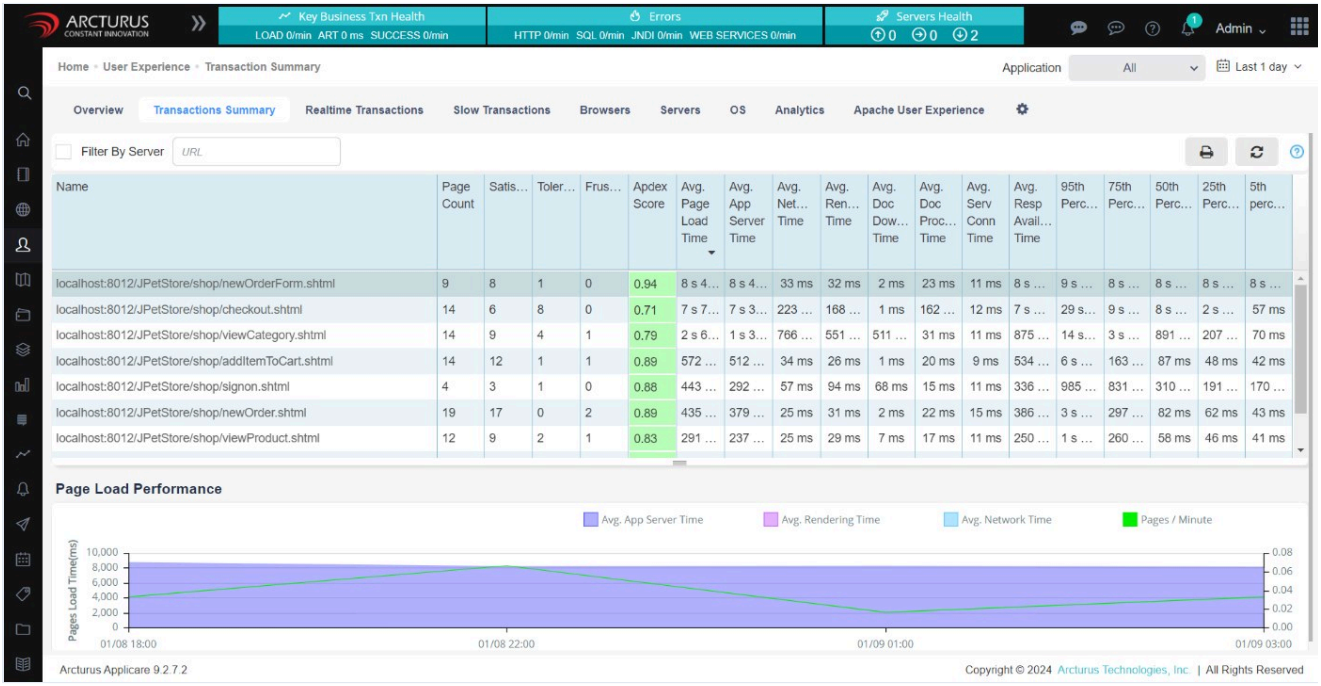
The End User Experience Analyzer presents historical performance data across various aspects in separate tabs. The Real-time Transactions tab updates immediately as users interact with your application, while data on other tabs may take a few minutes to appear.

Overview



The world map on the dashboard shows the performance breakdown for each country. Mouse over a country to see performance summary data for the country. On the right you can view top transactions by different criteria such as page count, average page load time etc... The chart below shows the performance data over the selected time period and if you click on a country in the map, the chart will display the performance data for the country as shown in the image below.

Transaction Summary



The Transaction Summary view shows performance statistics for transactions over a selected time period, with a timeline chart at the bottom. Clicking on a transaction will refresh the chart to display performance data for the selected transaction.

The **Apdex score** quantifies user satisfaction with web application performance on a scale from 0 to 1, where 1 signifies perfect satisfaction and 0 denotes complete dissatisfaction. It condenses various response times into a single metric.

By default, Applicare will consider the average response time of any transactions as its **Desired response time(T).**

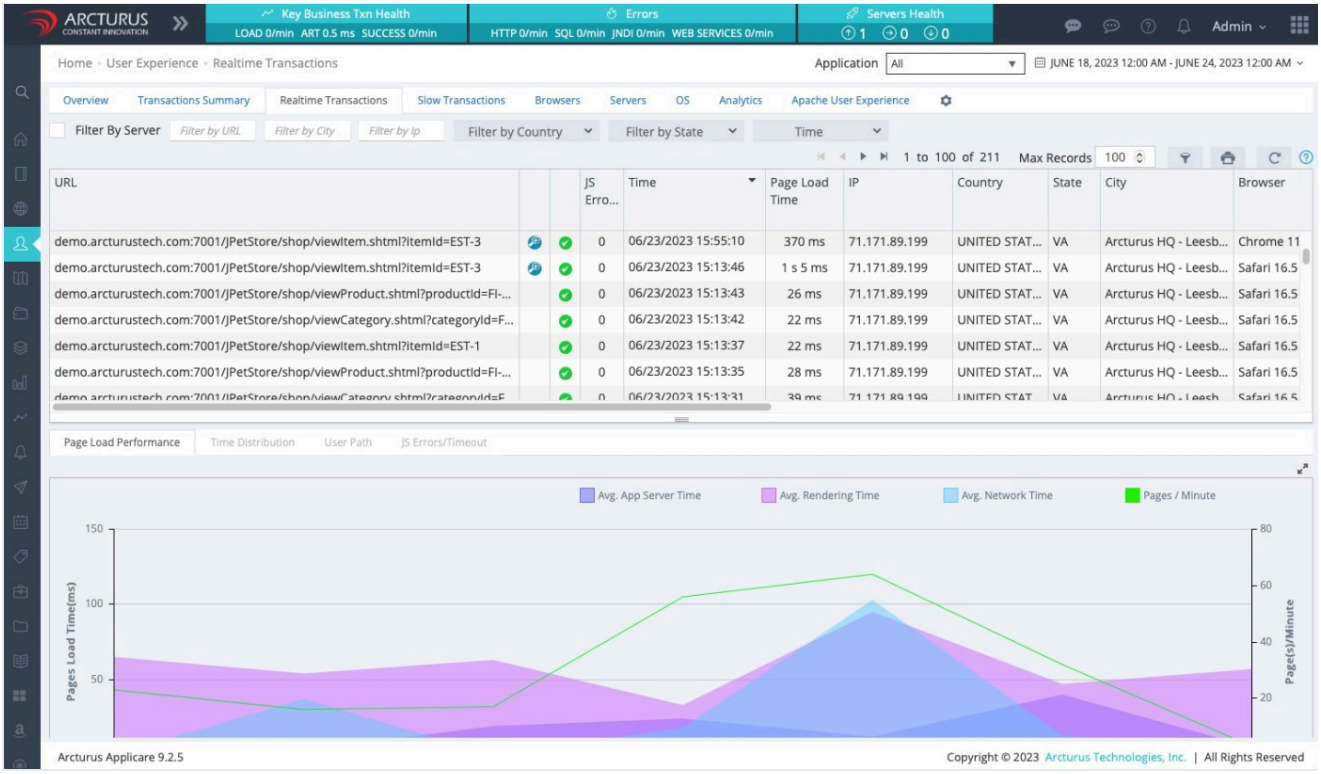
Transactions will be classified as follows:

- **Satisfied**: Response times ≤ T.
- **Tolerating**: Response times between T and 4T.
- **Frustrated**: Response times > 4T.

Apdex Score is calculated with the formula:

Apdex Score = (Satisfied Count + (Tolerating Count / 2)) / Total Count

Realtime Transactions



The Realtime view displays the last n number individual transactions recorded. Along with performance stats you can view location and browser information of users accessing your application. This view doesn't show any transactions older than a day.

In case the Applicare Controller/Collector collecting EUM data is behind a proxy, firewall or load balancer, make sure that the proxy, firewall or load balancing server forwards the client requests with "X-Forwarded-For" header parameter. If this parameter is not included in the HTTP request then Applicare may show the IP address of proxy, firewall or load balancing server in End User Experience Analyzer.

Below are the details on monitored data shown for each transaction

- **Resource Fetch Time:** The time it takes for a web browser to fetch a resource, such as an image, CSS file, or JavaScript file. This includes the time it takes to:
  - **DNS lookup:** The time it takes to resolve the domain name of the resource.
  - **TCP connection establishment:** The time it takes to establish a connection to the server that hosts the resource.
  - **TLS handshake:** The time it takes to establish a secure connection to the server, if the resource is hosted over HTTPS.
  - **Request:** The time it takes to send the request for the resource to the server.
  - **Response:** The time it takes to receive the response from the server, including the time it takes to download the resource.

    Resource Fetch Time is an important metric for measuring the performance of a web page. A short Resource Fetch Time means that the page will load more quickly, which can improve the user experience.

There are a number of factors that can affect Resource Fetch Time, including:

- **The size of the resource:** Larger resources will take longer to fetch.
- **The speed of the user's internet connection:** A slower internet connection will result in a longer Resource Fetch Time.
- **The performance of the browser:** A faster browser will be able to fetch resources more quickly.
- **The location of the server:** The closer the server is to the user, the shorter the Resource Fetch Time will be.

Here are some tips for improving Resource Fetch Time:

- **Use minified and compressed resources:** Minified and compressed resources are smaller, which means they will take less time to fetch.
- **Use a CDN:** A CDN (Content Delivery Network) can deliver resources from servers that are closer to the user, which can improve Resource Fetch Time.
- **Use a caching mechanism:** A caching mechanism can store resources that have already been fetched, which can reduce the need to fetch them again.

- **Document Download Time:** The time it takes to download a document from a web server to a user's device. This includes the time it takes to:
  - **DNS lookup:** The time it takes to resolve the domain name of the web server.
  - **TCP connection establishment:** The time it takes to establish a connection to the web server.
  - **TLS handshake:** The time it takes to establish a secure connection to the web server, if the request is made over HTTPS.
  - **Request:** The time it takes to send the request for the document to the web server.
  - **Response:** The time it takes to receive the response from the web server, including the time it takes to download the document.
- **Document Processing Time:** The time it takes for the browser to finish parsing the HTML code of a web page and construct the Document Object Model (DOM). The DOM is a hierarchical representation of the page's elements, which can be manipulated by JavaScript.

  Factors Affecting Document Processing Time:
  - **HTML Code Size**: Larger HTML files take longer to parse.
  - **HTML Code Complexity**: More complex HTML structures require additional parsing time.
  - **Browser Performance**: Faster browsers can parse HTML more quickly.
  - **JavaScript Presence**: JavaScript files can increase document processing time.

  Tips for Improving Document Processing Time:
  - **Use minified and compressed HTML code-** Minified and compressed HTML code is smaller, which means it will take less time to parse.
  - **Use a CDN-** A CDN (Content Delivery Network) can deliver HTML code from servers that are closer to the user, which can improve document processing time.
  - **Use a caching mechanism-** A caching mechanism can store HTML code that has already been parsed, which can reduce the need to parse it again.

  - **Optimize the JavaScript files-** Optimizing JavaScript files can improve their performance and reduce the document processing time.

- **Time to First Byte (TTFB)-** A metric that measures the time it takes for a web browser to receive the first byte of a response from a web server. This includes the time it takes to:
  - **DNS lookup-** The time it takes to resolve the domain name of the web server.
  - **TCP connection establishment-** The time it takes to establish a connection to the web server.
  - **TLS handshake-** The time it takes to establish a secure connection to the web server, if the request is made over HTTPS.
  - **Request-** The time it takes to send the request for the resource to the web server.
- **DOM Ready Time:**

  The duration it takes for a web browser to parse the HTML code and build the Document Object Model (DOM) of a web page. The DOM represents the page's elements in a hierarchical structure, which can be manipulated by JavaScript. The DOM Ready Time event is triggered when the DOM is ready, though the page may still be loading images, CSS, and other resources. This metric indicates how quickly a page becomes interactive.

  Factors affecting DOM Ready Time include HTML code size and complexity, the user's internet speed, and browser performance.

  Improving DOM Ready Time can enhance user experience by making the page interactive more quickly. Here are some tips:

  - Minify and compress HTML, CSS, and JavaScript files.
  - Reduce the number of large images or resources.
  - Utilize a Content Delivery Network (CDN) for static resources.
  - Implement caching to store frequently accessed resources.
- **Network Time-** The duration for a web browser to retrieve all resources required to load a web page, including:
  - **DNS lookup-** Time to resolve the domain name of the resources.
  - **TCP connection establishment-** Time to set up a connection with the resource servers.
  - **TLS handshake-** Time to establish a secure connection for HTTPS resources.
  - **Request-** Time to send requests to the servers for the resources.
  - **Response-** Time to receive responses and download the resources.
- **Rendering Time-** The time needed for a web browser to display a web page, including:
  - **Parse the HTML code:** Time to parse HTML code and build the DOM.
  - **Load the CSS and JavaScript files:** Time to load CSS and JavaScript files for styling and interaction.
  - **Render the images and other resources:** Time to render images and other resources used on the page.

There are a several reasons why DOM ready time might be high while document processing time can remains low.

- **The HTML code is large and complex.** A large and complex HTML code can take longer to parse, even if it does not contain any JavaScript. This is because the browser has to process more information to create the DOM.
- **The web page uses a lot of third-party resources.** Third-party resources, such as images, CSS, and JavaScript files, can also add to the DOM ready time. This is because the browser has to wait for these resources to load before it can create the DOM.
- **The web page is using a slow web server.** If the web server is slow, it can take longer for the browser to download the HTML code and third-party resources. This can also increase the DOM ready time.

If your DOM ready time is high, but your document processing time is low, then you may need to optimize your HTML code to make it smaller and easier to parse.

If you access Realtime Transaction and double click on any URL you may encounter a message "Trace not available..."

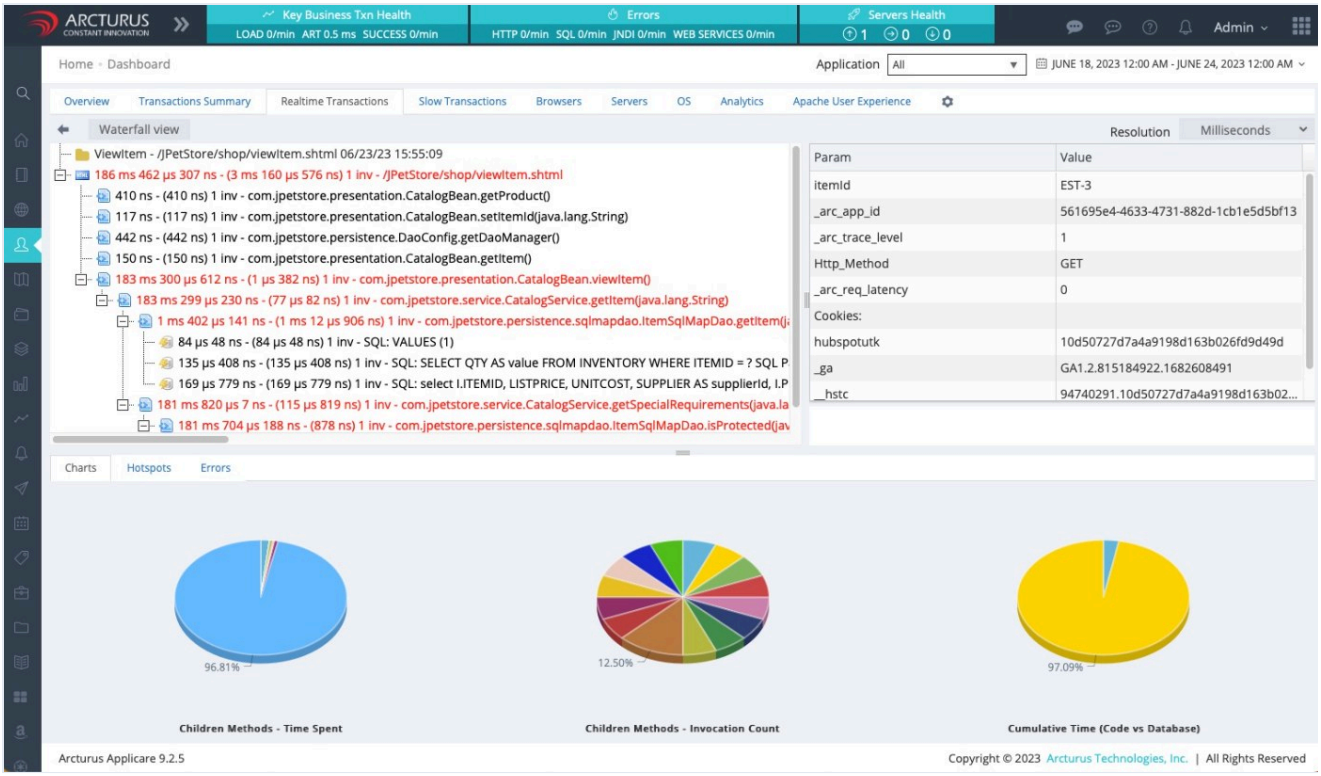
Activating tracing on End User Experience

1. Navigate to **Server Analyzer > IntelliTrace > HTTP**
2. Right-click the URL you want to trace in detail tracing and select "Add as a Key Business Transaction".
3. Add the URL to Key Business Transaction. Alternatively, go to directly visit **Business Transactions > Configure Key Business Transactions** and include the URL to Business Transaction Monitoring.
4. Go to **Profiling > Configure Business Transactions > Transaction Monitoring**.
5. Activate Per-Transaction Monitoring.
6. Activate Transaction Tracing.

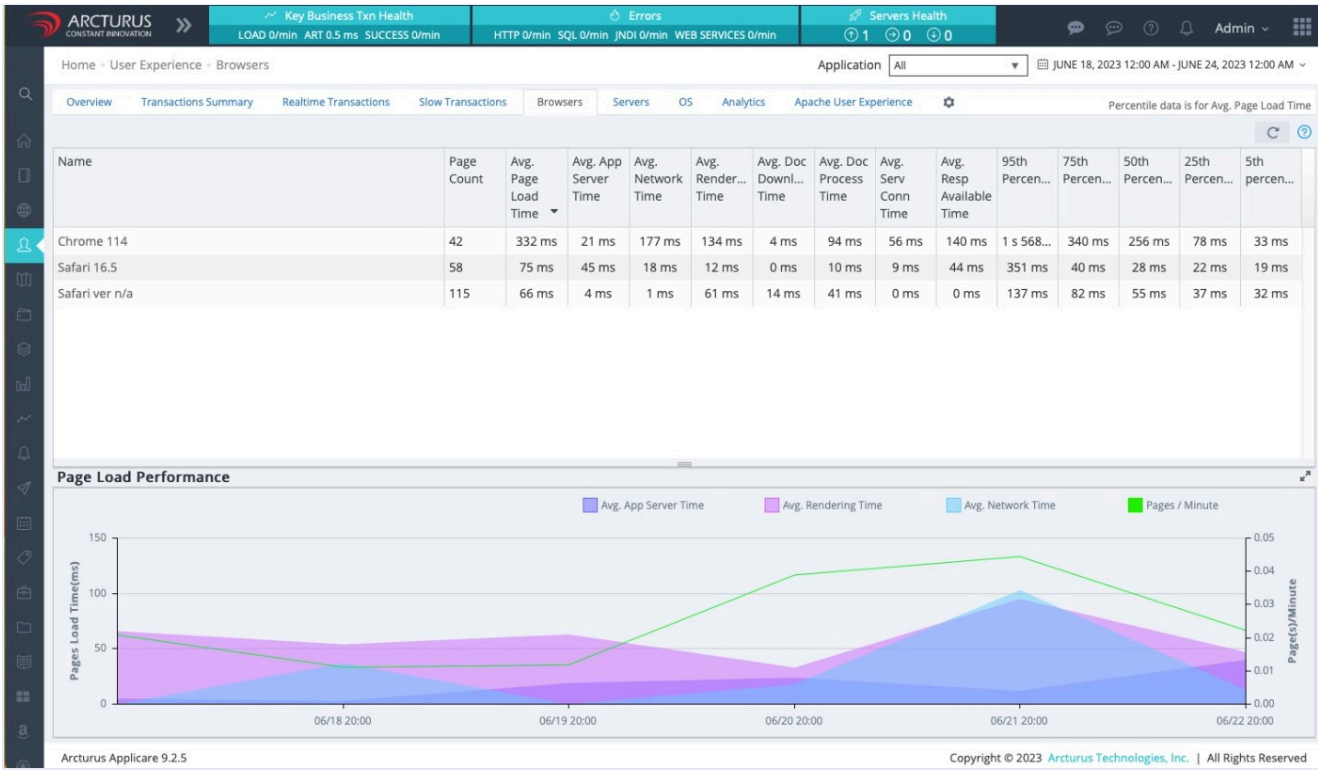
Once you have followed the above steps, restart the agent jvm.

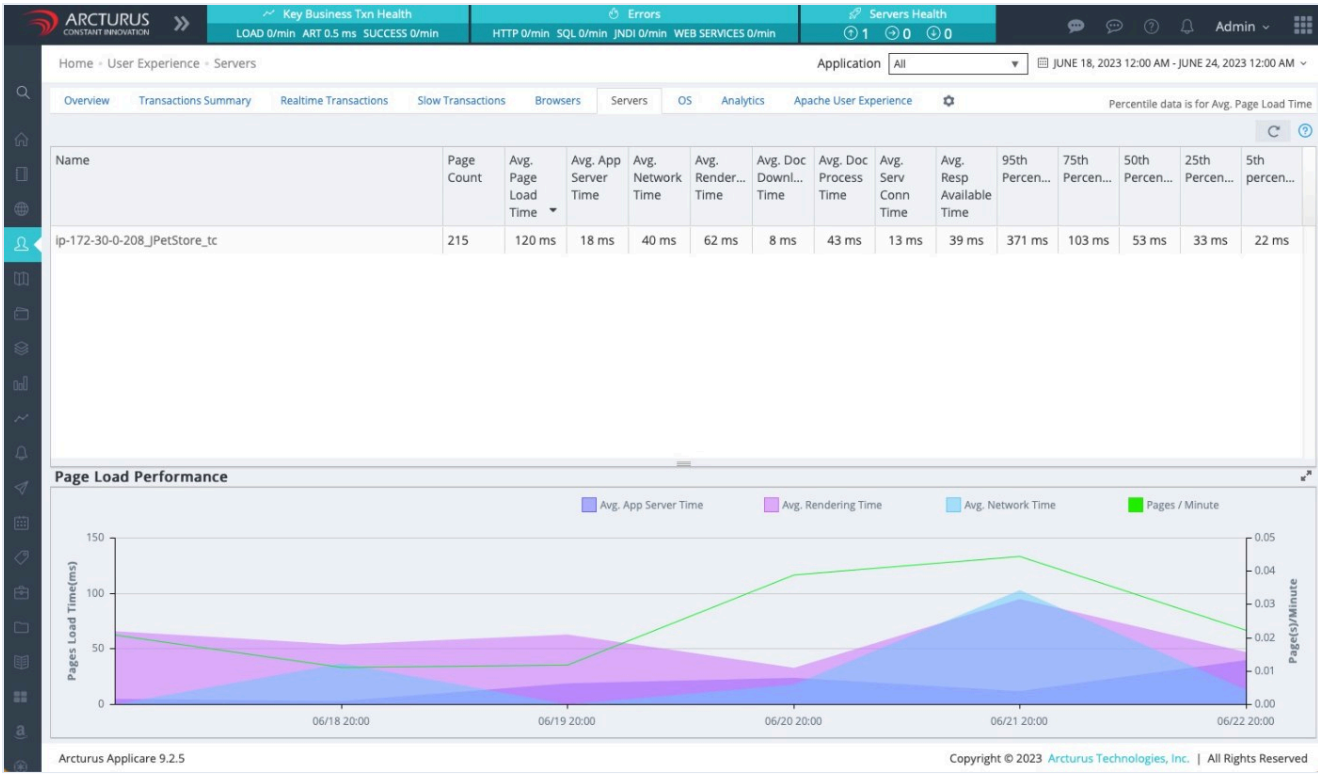
Now you can visit the UEM > End User Experience Analyzer > Realtime Transactions and double click on the URL.

If you have added the URL as KBT, detailed trace will be displayed as shown in the following image.



Browser and Backend Server Views





The Browsers and Servers tabs are similar to the Transaction Summary view but provide performance data segmented by client browser type and backend servers handling transactions.

End Client Processes Monitoring

End Client Process Monitoring tracks the running processes and resources utilized by each process on the end client machine where browser is running. This helps identify if the slowness is caused by some process running on End User's computer. End Client Process Monitoring is only supported on Windows and Internet Explorer (IE) end clients. End Client Process Monitoring requires permissions from the end user (or centrally configured by administrators) to allow running 'Microsoft Script Runtime'. If these permissions are not centrally configured then users are prompted on first visit to the pages with end client process monitoring enabled.

End Client Process Monitoring can be enabled for slow calls or all calls.