USING APPLICARE › INFRASTRUCTURE

# Server Analyzer

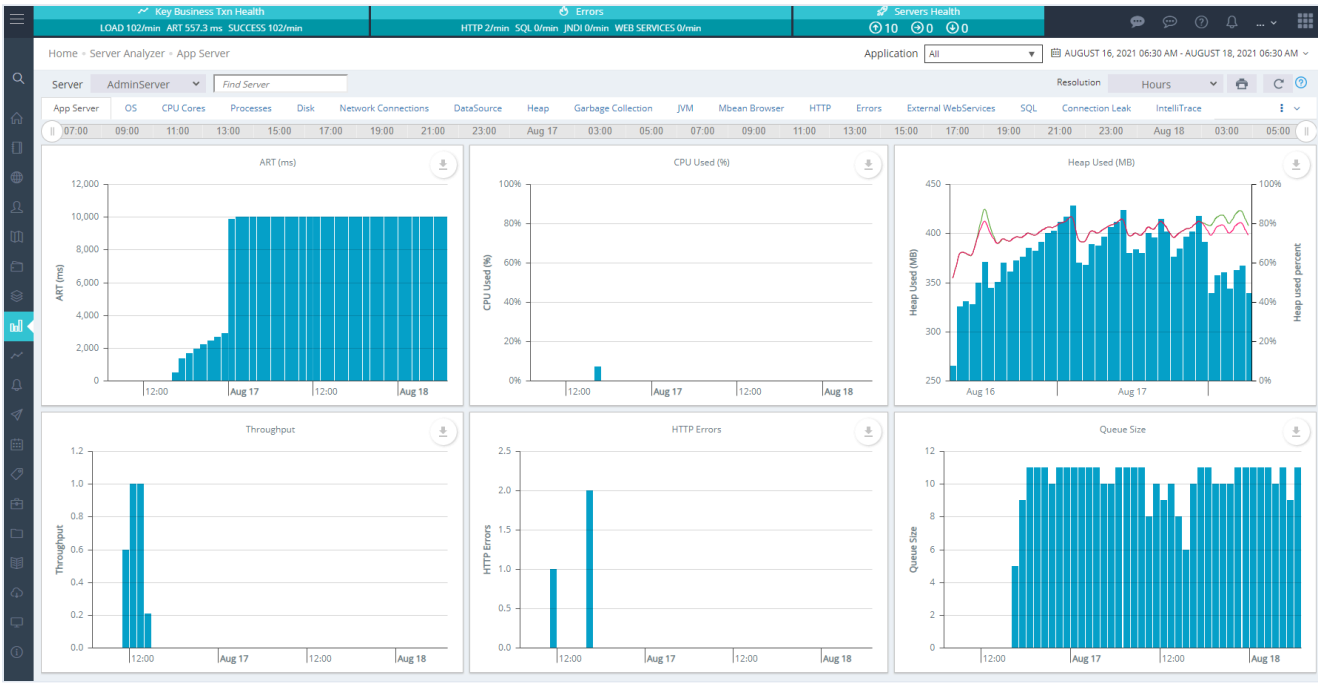
Applicare 10.0 · Updated April 5, 2025 · **Pending technical review**

## Overview

The Server Analyzer gives you a quick look at your server's health and activity. Select a time frame at the top to view historical data. The server list will show only those related to chosen application. To zoom in, click and drag over the area you want to explore.

### App Server Analyzer

The AppServer tab offers a quick overview of application server performance. It helps identify potential bottleneck, such as queue buildup, which indicates the server is struggling to process incoming request. Issues like garbage collection or high CPU can also be mentioned here. The AppServer Analyzer provides a correlated view to pin point the effects of change in performance metrics.




If a time period is selected on top of the home dash then history view is visible. Average Response Time (ART) is a great indicator of a server's performance. A significant increase in ART means things are getting stuck somewhere in the app server. Similarly heap used, throughput, queue size, http errors and CPU used are available.

**Throughput**: Server throughput is the number of requests served per second by the server. **Busy Threads**: The number of Server threads thats are busy serving requests at any given time.

### OS Analyzer

The OS Internals Analyzer provides a fast and convenient window into your physical server's/VM's performance including # of processes, tcp connections, CPU, memory and storage performance.

**Processes:** Number of processes running on the machine. Running too many processes may cause memory contention (paging), I/O contention or excessive context switching. This contention can reduce system throughput if parallel execution were not used.

**CPU %-** Cpu usage in percent. High CPU usage is not a significant cause for concern as long as it doesn't occur over the long-term. But if the cpu usage remains high for long term then they system may freeze or will respond too slow.

**Process Queue Length-** The Processor Queue Length is the number of threads that are ready but currently unable to run on the processor due to another active thread. This count shows ready threads only, not threads that are running. A bottleneck on the processor may be thought to occur where the number of threads in the queue is more than 2 times the number of processor cores over a continuous period.

**Memory Free %-** Percentage of free memory available in physical memory. Memory can have a huge impact on a system is performance. As physical free space increases, the machineï¿½s dependency on virtual memory decreases. The less a computer has to depend on virtual memory, the more efficiently that machine will run.

**Memory Free-** Free memory available (in Mega Bytes) in physical memory.

**Page File Usage %-** Displays the percentage of the paging file that is currently in use.

**Disk Transfers/sec-** A high Disk Transfers/sec value may occur due to a burst of disk transfer requests by either an operating system or application.

**Context Switches-** This attribute reports the combined rate at which all processors on the computer are switched from one thread to another. High context switches, shows that threads have less time to do their work and the system performance might go down.

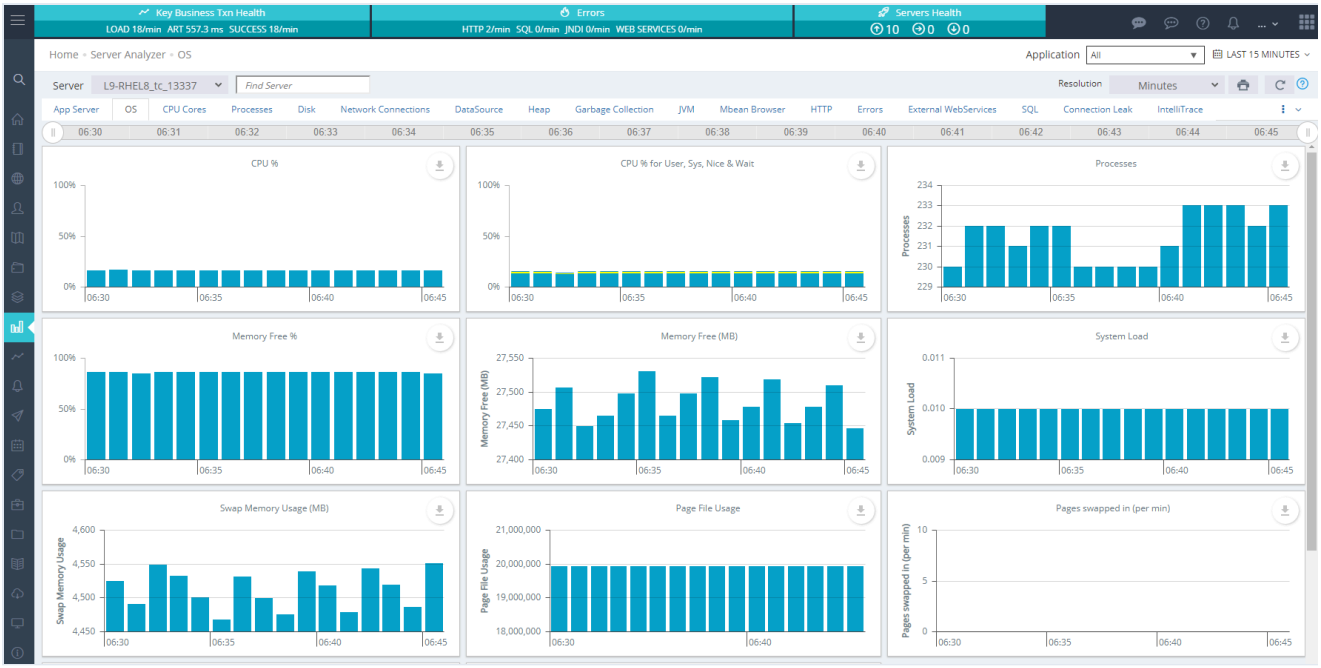
**Swap memory usage-** Swap usage refers to the amount of virtual memory that is currently being used to temporarily store inactive pages from the main physical memory. Swap space is your safety net for when you run out of RAM. While swapping, processes may get slower, data is still nevertheless processed. However, when you run out of virtual memory, processes are queued and stalled until some memory is freed up.

**Pages Swapped In-** Number of pages read(in last one minute) from paging/swap file(s) to resolve hard page faults. (Hard page faults occur when a process requires code or data that is not in physical memory, and must be retrieved from disk.)

**Pages Swapped Out-** Number of pages written(in last one minute) to paging/swap file(s) to free up space in physical memory. Pages are written back to disk only if they are changed in physical memory, so they are likely to hold data, not code. A high rate of pages output might indicate a memory shortage. The operating system writes more pages back to disk to free up space when physical memory is in short supply. Even when swap space is not particularly full, continuously high swap-out rates indicate that the pages that are swapped are actively processed. This means that your system requires more fast physical memory for your workloads, and swap is not helping.

**TCP Connections-** Total TCP connections count.

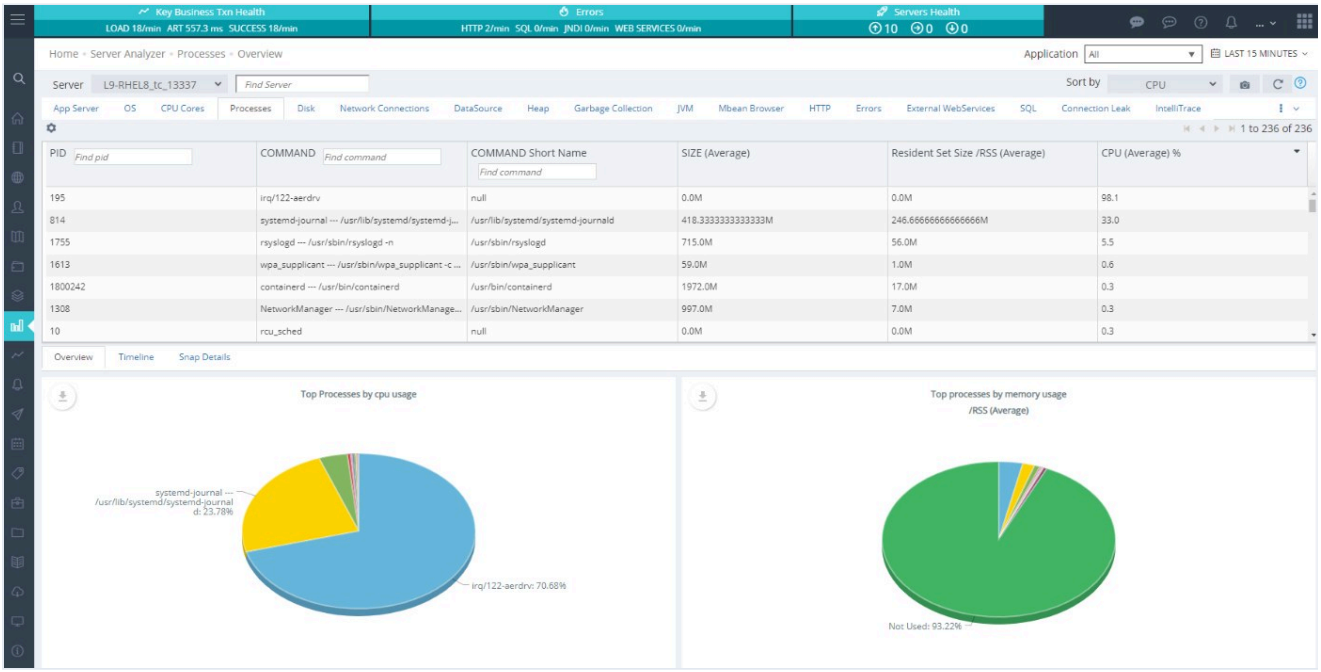
Here, you can access the OS performance history for the selected time period.




## Process Analyzer

This view displays the performance stats for CPU, memory and running processes organized in somewhat similar manner to the output of **Top** utility on Unix based systems.

snapshots of various processes running at any point of time and what kind of resources they were consuming at that point of time is available here. Applicare by default would take a snapshot of this info every 30 minutes but this can be configured by changing the snaps interval.



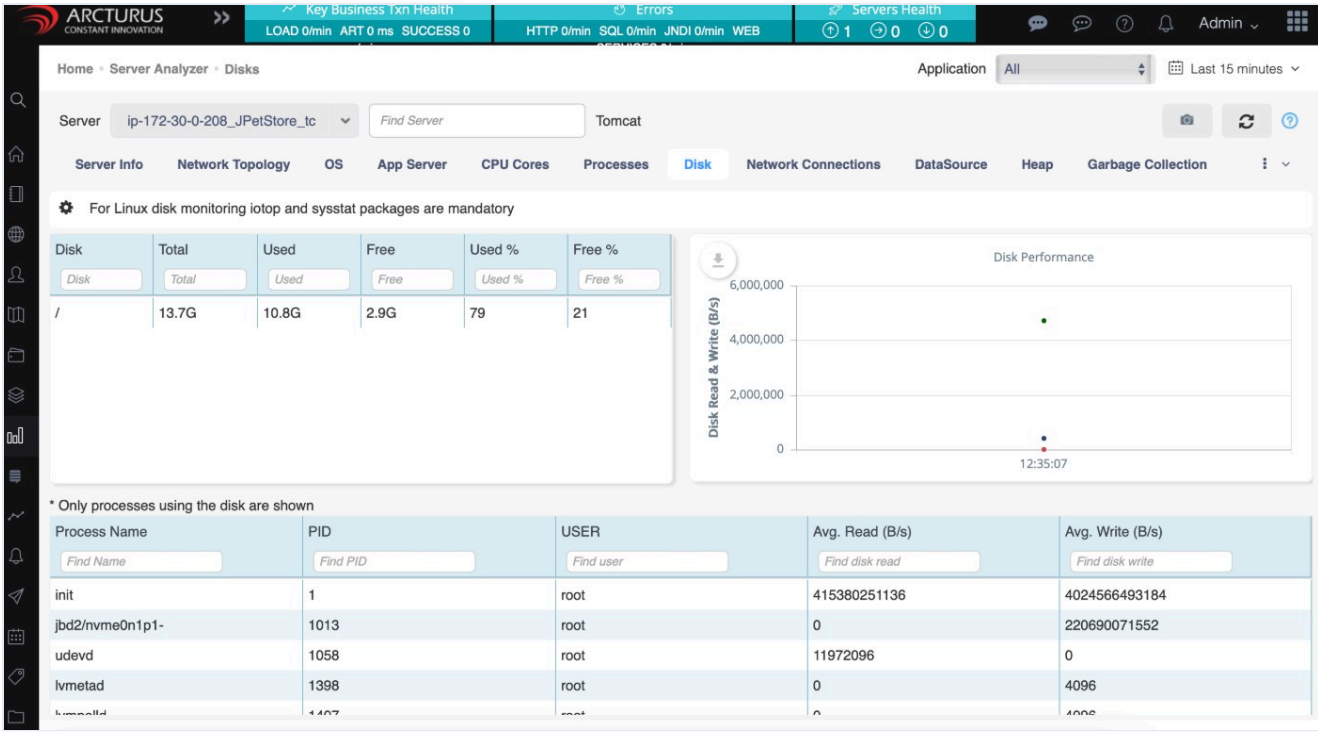

### Column Descriptions:

| | |
|---|---|
| **PID :** | **The process ID of each task.** |
| **USER :** | The user name of the task's owner. |

| | |
|---|---|
| **PID :** | **The process ID of each task.** |
| **STIME :** | Time the process was started. |
| **SIZE :** | The size of the task's code plus data plus stack space, in megabytes, is shown here. |
| **RSS :** | The total amount of physical memory used by the task, in megabytes, is shown here. |
| **SHARE :** | The amount of shared memory used by the task is shown in this column. |
| **STATE :** | The state of the task is shown here. The state is either S for sleeping, D for uninterruptible sleep, R for running, Z for zombies, or T for stopped or traced. |
| **TIME :** | Total CPU time the task has used since it started in MM:SS format. |
| **CPU :** | The task's current usage of CPU time as a percentage. |

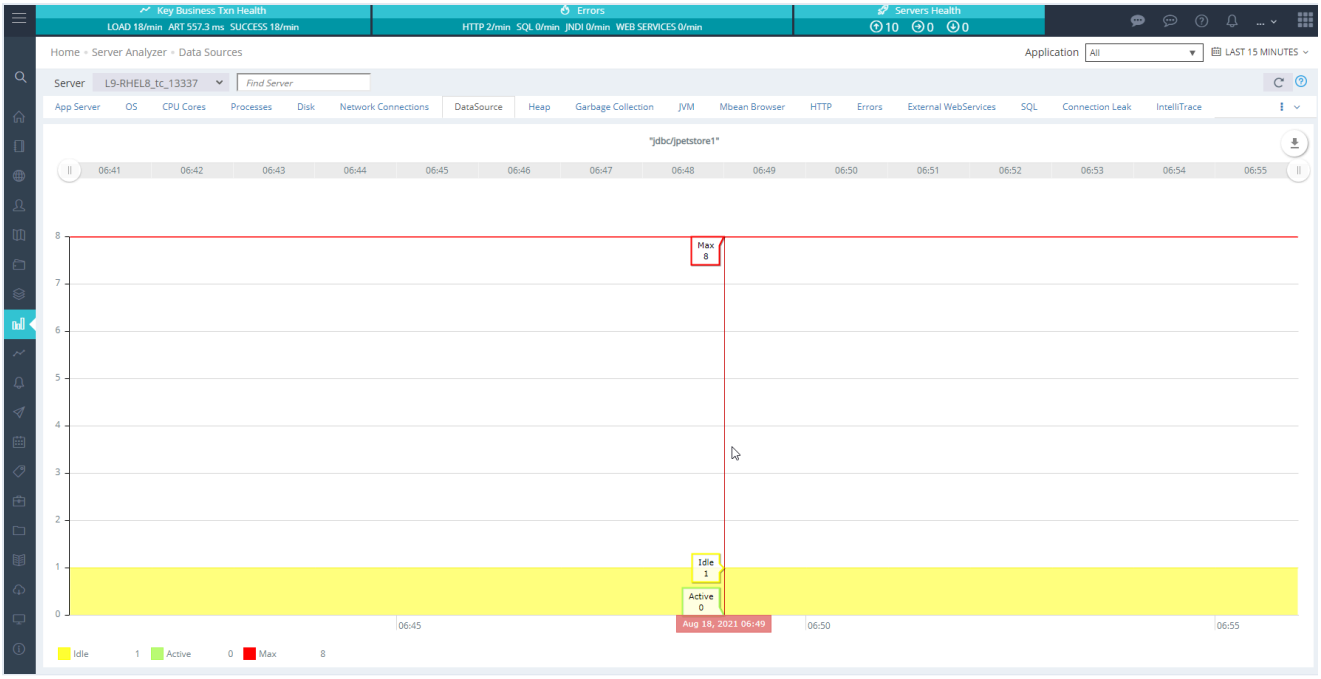
## DISK

On the left side, view the used and available disk space. On the right side, monitor disk performance, with only disk-using processes shown.




## Data Source Analyzer

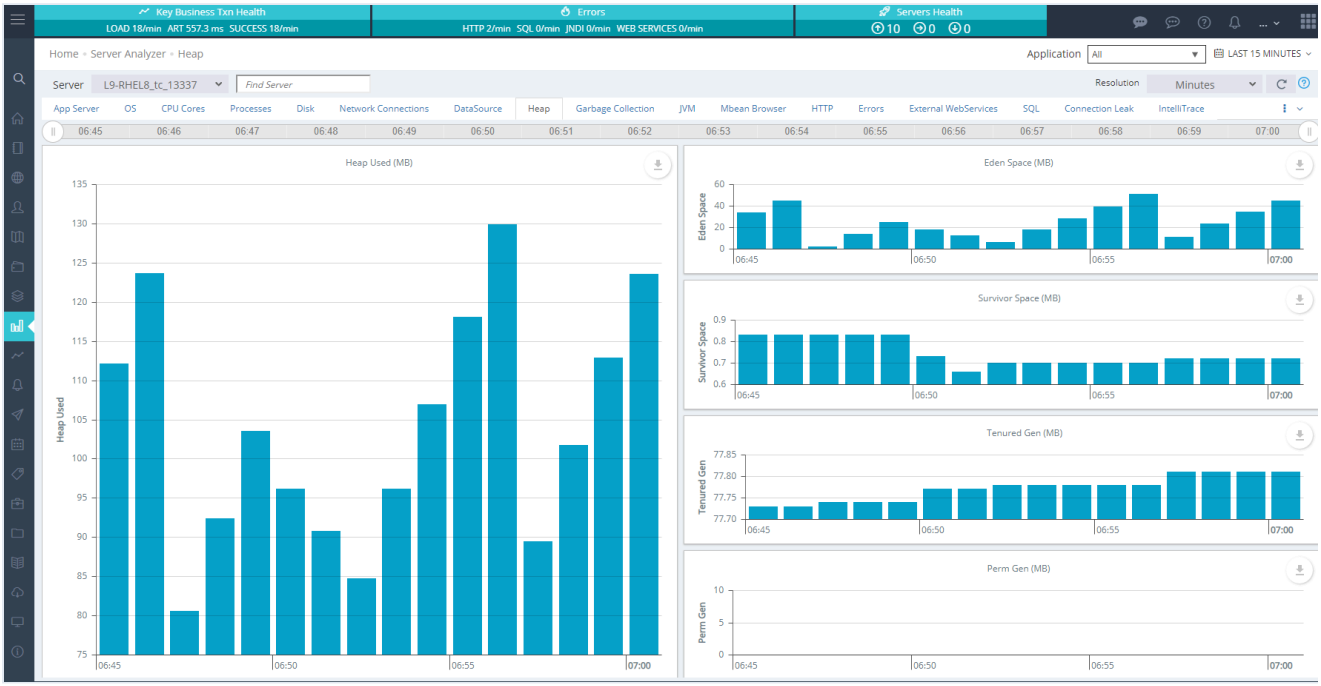
The DataSource Analyzer provides a fast and convenient window into various data source's usage. Datasource utilization is very useful in finding out issues related to connections running out and tuning of connection pools.




## Heap Analyzer

The Heap Analyzer provides a convenient window into the history of your server's garbage collection and memory allocation behavior. You can zoom in and out, choose the range of days from the drop down, and select which server to display.

You can easily spot slow memory leaks with the Heap Analyzer. Zoom out to show the data for as long as the server has been up, and you can see how each generation is growing over time. If after major garbage collections the heap doesn't settle to prior levels, but instead keeps increasing, you may have a memory leak.



### Heap Dump

You can generate a heap dump of the selected server by clicking on Heap Dump button. Please note this will make the server unresponsive until the heap dump is written to a file on the server. Generating heap dumps on demand is availble on

- Sun JVM 6 and above. - Oracle JRockit R28 and above.

Also you can enable automatic heap dumps on supported JVMs when available free memory reaches a certain threshold by setting 2 startup arguments.

e.g. Using startup arguments *-Dapplicare.dumpheap.perc=10 -Dapplicare.dumpheap.maxcount=2* will generate heap dumps when the server's available free heap size reaches 10% of the max heap size and will create maximum of 2 heap dumps during the lifetime of the JVM.

You can use the generated heap dumps to detect memory leaks and analyze the objects and their relationships by loading the heap dump in Arcturus Memory Analyzer.

## Garbage Collection Analyzer

The GC Analyzer provides a fast and convenient window into your server's garbage collector behavior. This gives you the ability to quickly spot garbage collection issues and tweak GC parameters to improve performance and reduce latency of your applications.

GC Analyzer is available on Oracle Sun JVMs, JRockit and IBM JVMs. It requires verbose GC enabled on the JVM.

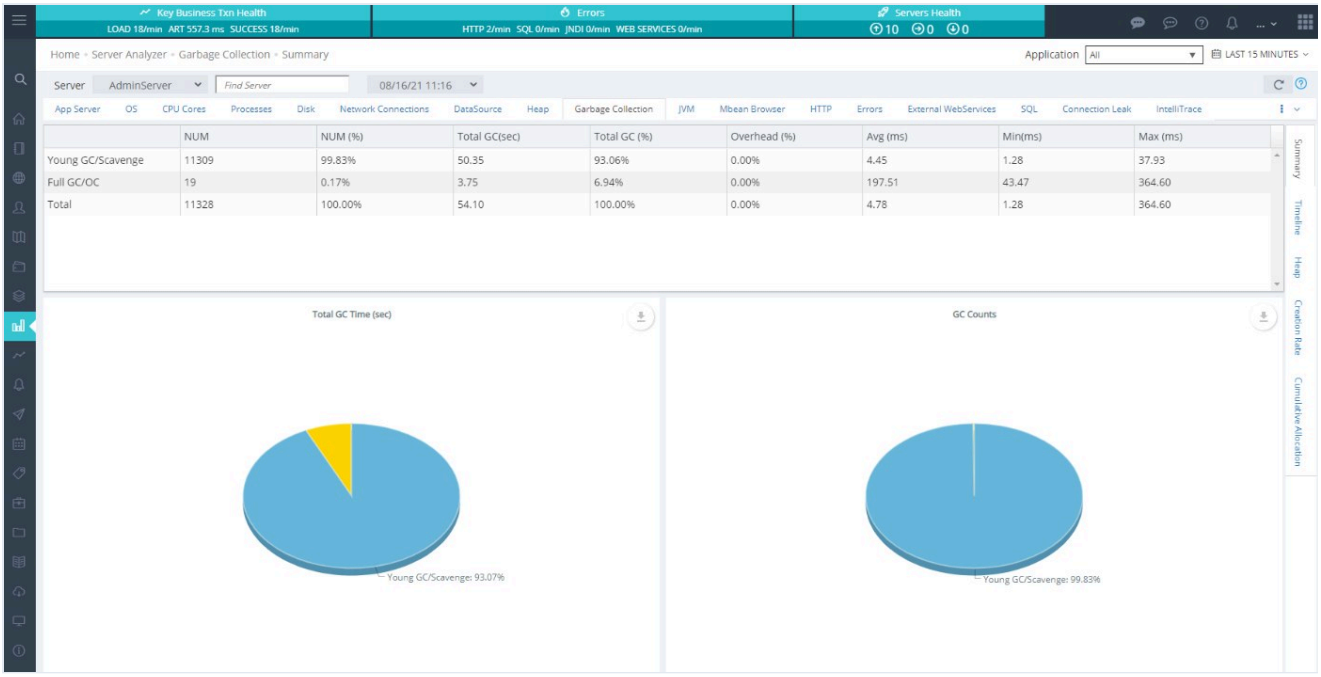
e.g. following arguments are automatically added by Applicare to enable verbose GC on Oracle Sun JVM.

**-verbosegc -Xloggc:%SERVER_NAME%.gc**

Also the use of **-XX:+PrintTenuringDistribution** flag is not supported at the moment and the use of it will prevent Applicare from reading the GC log file.

### GC Summary

The summary tab gives you an overview of Full and Young GC cycles that occured over the lifetime of JVM. An important value you should take note of is the Overhead (%) in the table.

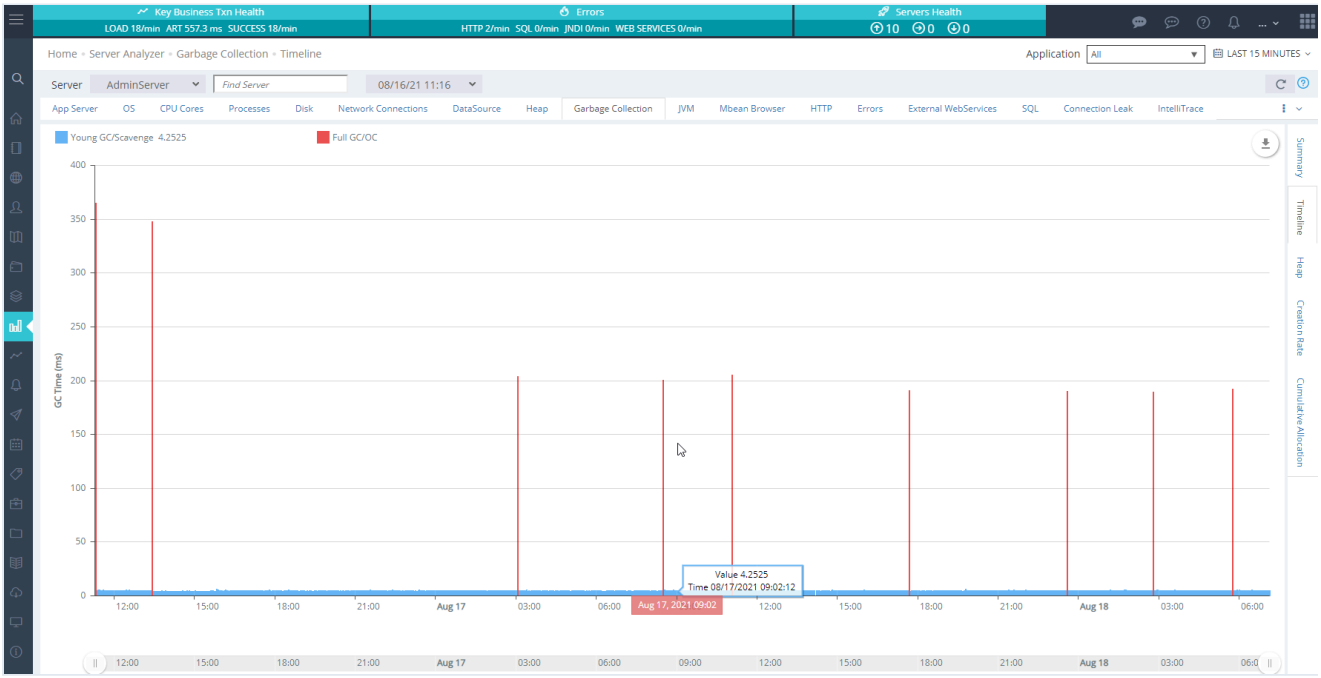



## Detailed Charts

The GC Analyzer feature charts that visually represent various aspects of Garbage Collection cycles.

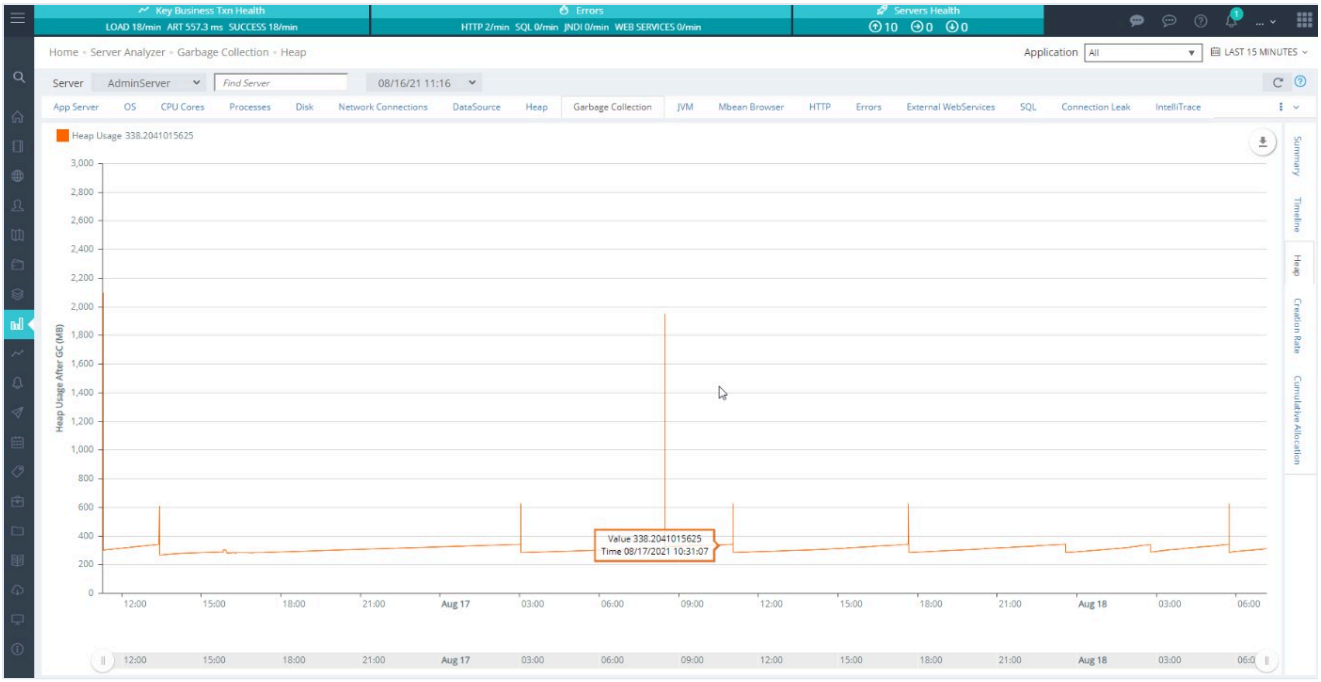
### GC Timeline

Garbage Collection shows each GC cycle over time and provides a clear picture of the type of GC cycle and time taken by each.



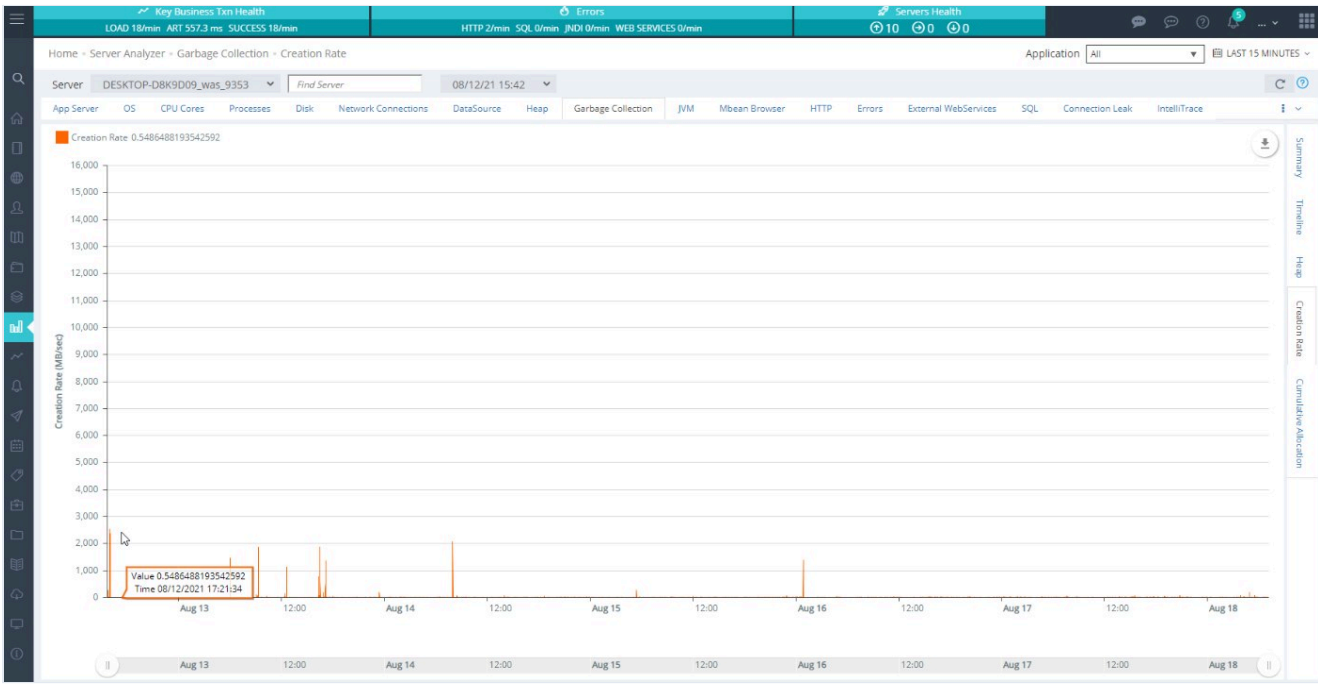

### Heap

This chart plots the Heap size before and at the end of each GC cycle over time.



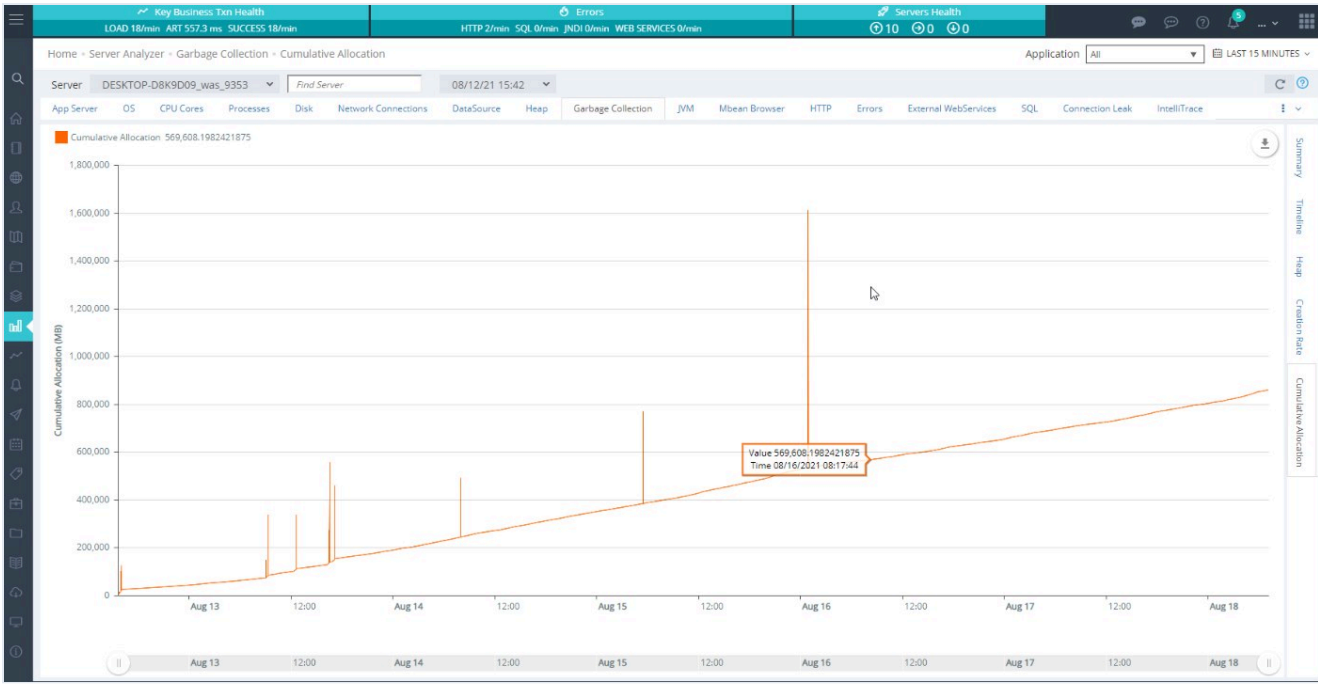

## Creation Rate

This analyzer shows the total size of objects created between each GC cycle.




## Cumulative Allocation

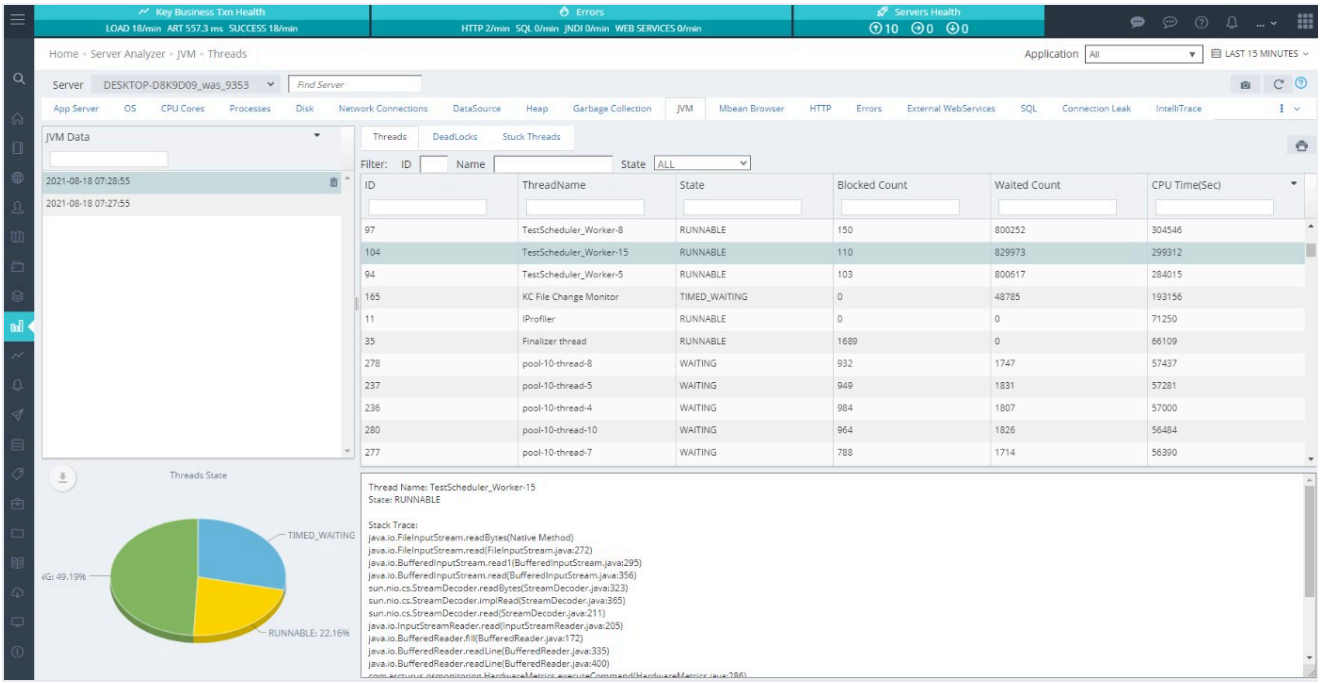
This analyzer shows total size of objects created over time.




## JVM Analyzer

The JVM Analyzer provides a very detailed view into a server's thread behavior, detects Deadlocks, detects stuck threads, and provides an interface for viewing stack traces.

This view allows you to save a thread dump of a running server and analyze the saved dumps at a later time. The saved snap shots contain all thread information displayed on the real time view and any deadlock or stuck threads detected at the time of creating the thread dump.



## Thread Analysis

### Deadlock Detection

A ***Deadlock*** is a situation where two or more threads are blocked forever, waiting on each other to finish. You can find a deadlock by taking a thread dump to display the current threads and look for the BLOCKED state, or you can use the **JVM Analyzer** and click on the **Deadlocks tab.** This tab shows all available information about the deadlock situation. When a deadlock occurs, the tab will turn red to alert you that a deadlock exists. Then you can capture the data and send it to development so they can analyze the code.

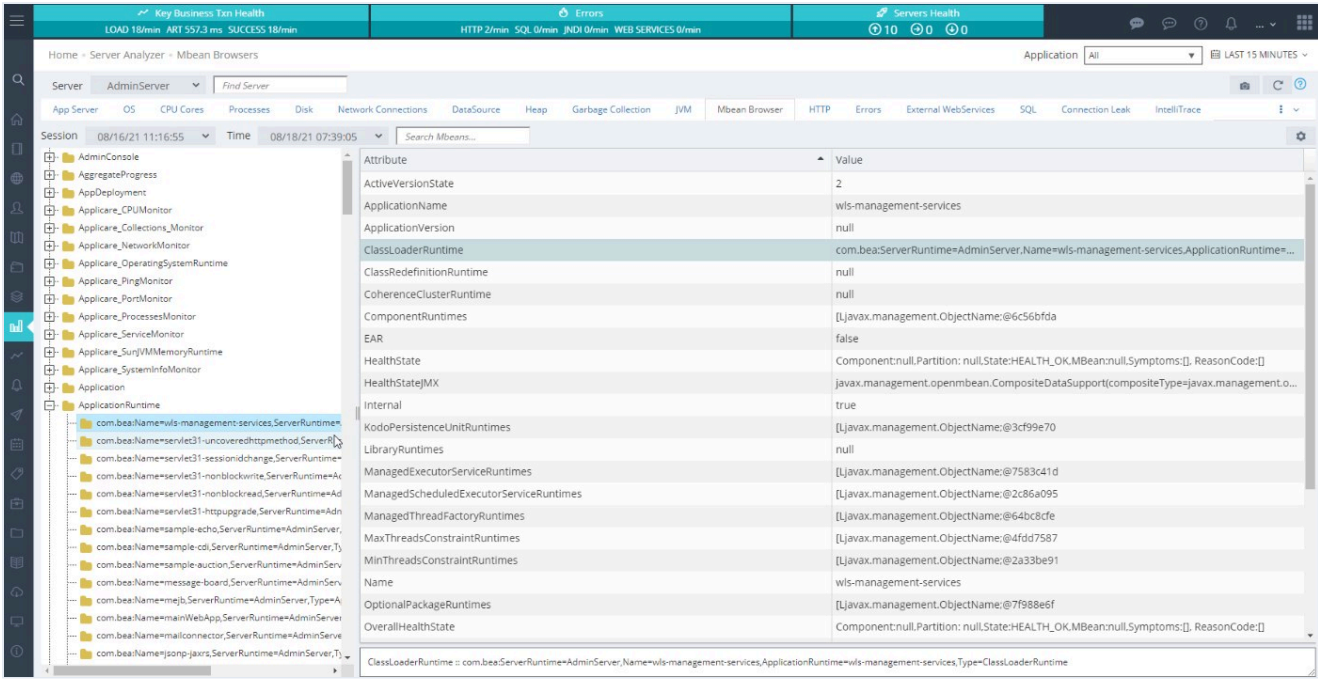
good course of action to follow when you detect a deadlock is to assign a portion of your team to concentrate on finding out how to reproduce the deadlock, and the other portion to look at the stack traces to figure out why the code is deadlocking

### Stuck Thread Detection

A ***stuck thread*** is a thread that is blocked and can't return to the thread pool within a defined period of time. When an application thread is blocked, it can't quickly finish its job and be reused. In most situations, the cause of stuck threads is also the cause of poor system performance. This is because the stuck thread interferes with regular task execution. The “Stuck Threads” tab will turn red and display the diagnostic information to help you identify the root cause of the stuck thread. Stuck Threads are often caused by the code opening a network connection without specifying a network connect or read timeout. When a timeout isn’t configured for each method call that involves networking, it will potentially create a block because it will wait infinitely for a response. The fallback timeout in this case comes from the OS which is by default set very long (e.g. 5-10 minutes). Especially in the case of SOA and Web Services where there are many network calls, timeouts need to be defined properly.

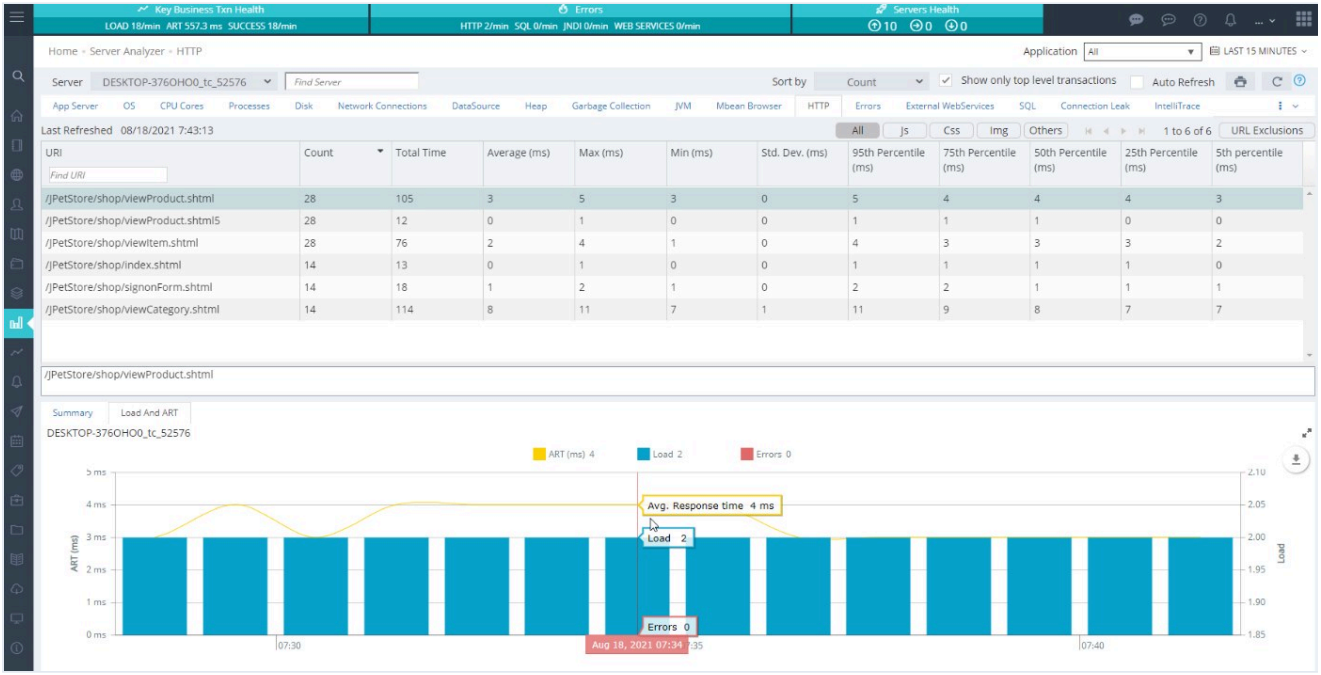
## MBean Browser

The MBean Browser enables you to explore MBean properties and their values.



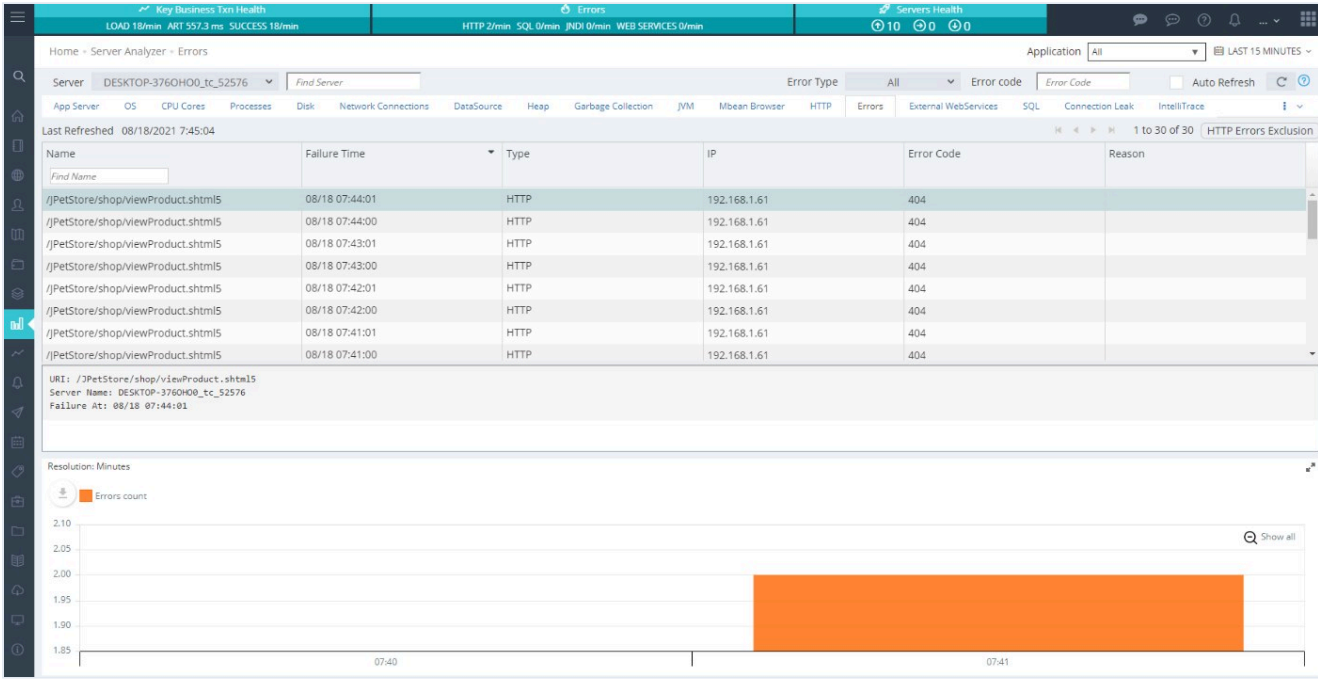
## HTTP

The HTTP Analyzer offers performance statistics for HTTP requests on the selected server. The real-time view displays HTTP stats since the JVM started, while the historical view provides data for the selected time period. The real-time view also indicates whether HTTP Profiling is enabled and, if so, how long it has been active.




## HTTP Errors

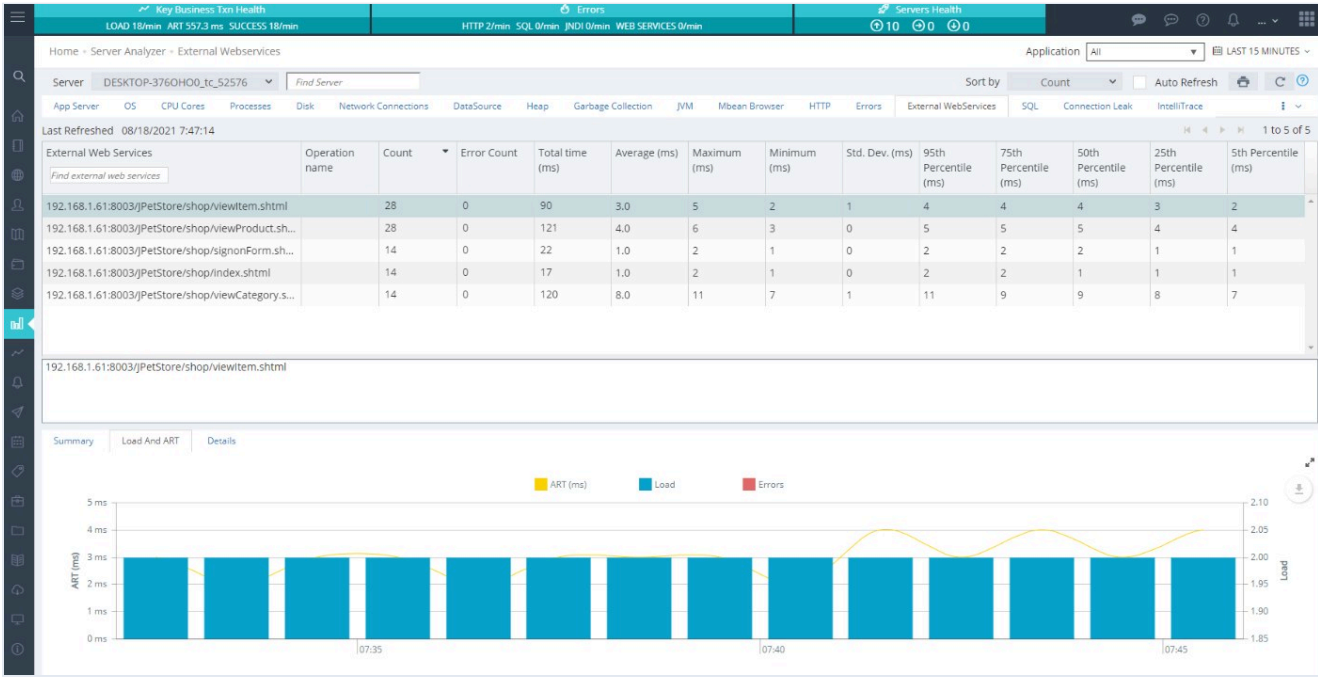
HTTP Errors shows URI, failure time, http error code, exception and IP address where the call came from.




## External WebServices

Server Analyzer External WebServices shows all the web services that are called from the monitored JVM/App Server.

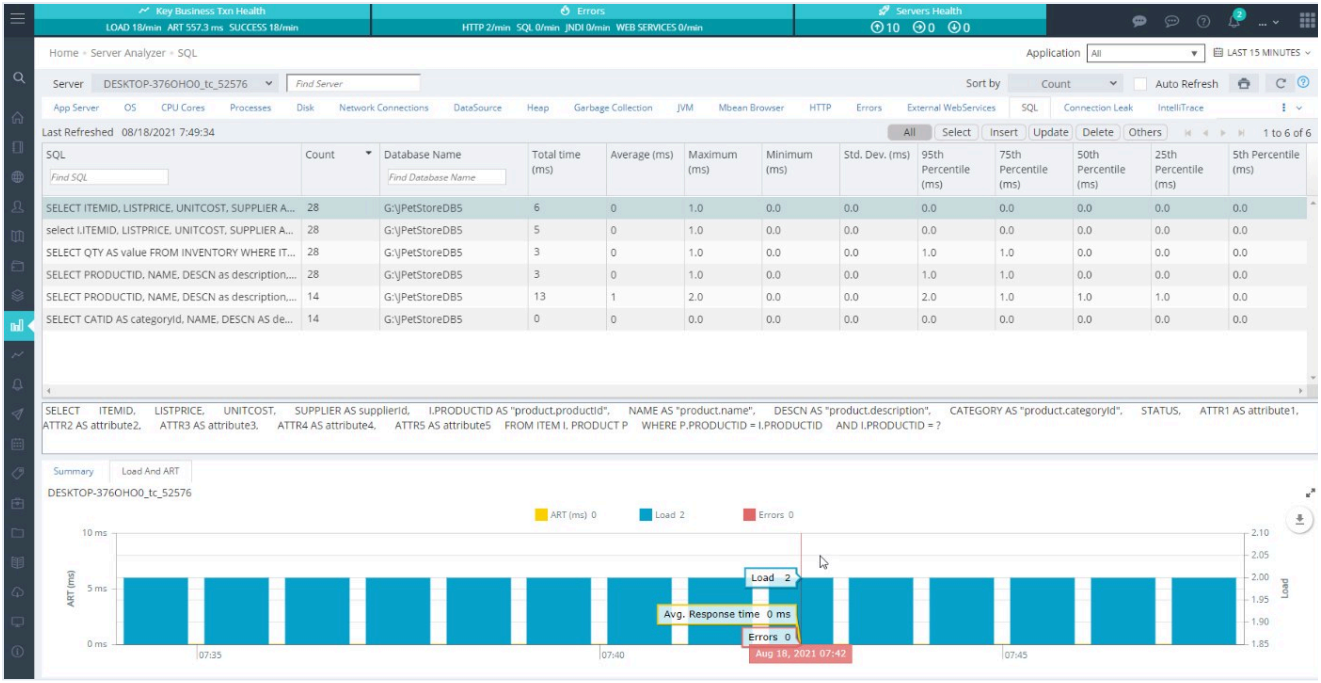
External WebServices monitors all calls to externally hosted webservices and provides performance stats, server ip and port where the calls are going for easily tracking the issues that come with external webservices response time going up.



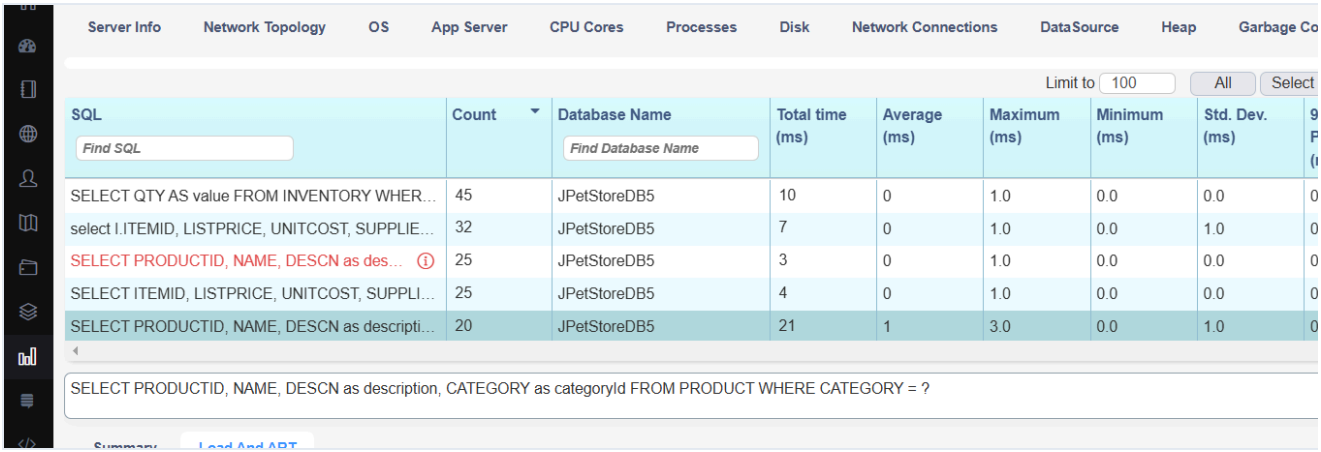

## SQL

IntelliTrace Analyzer contains separate views to display data collected by IntelliTrace aspects. SQL Statement Profiling provides real-time performance statistics of database requests. You can view what statements are taking the most time and quickly diagnose performance impacting SQL/s.

Maximum (ms) and Minimum (ms) data will not be available for database monitoring agents.

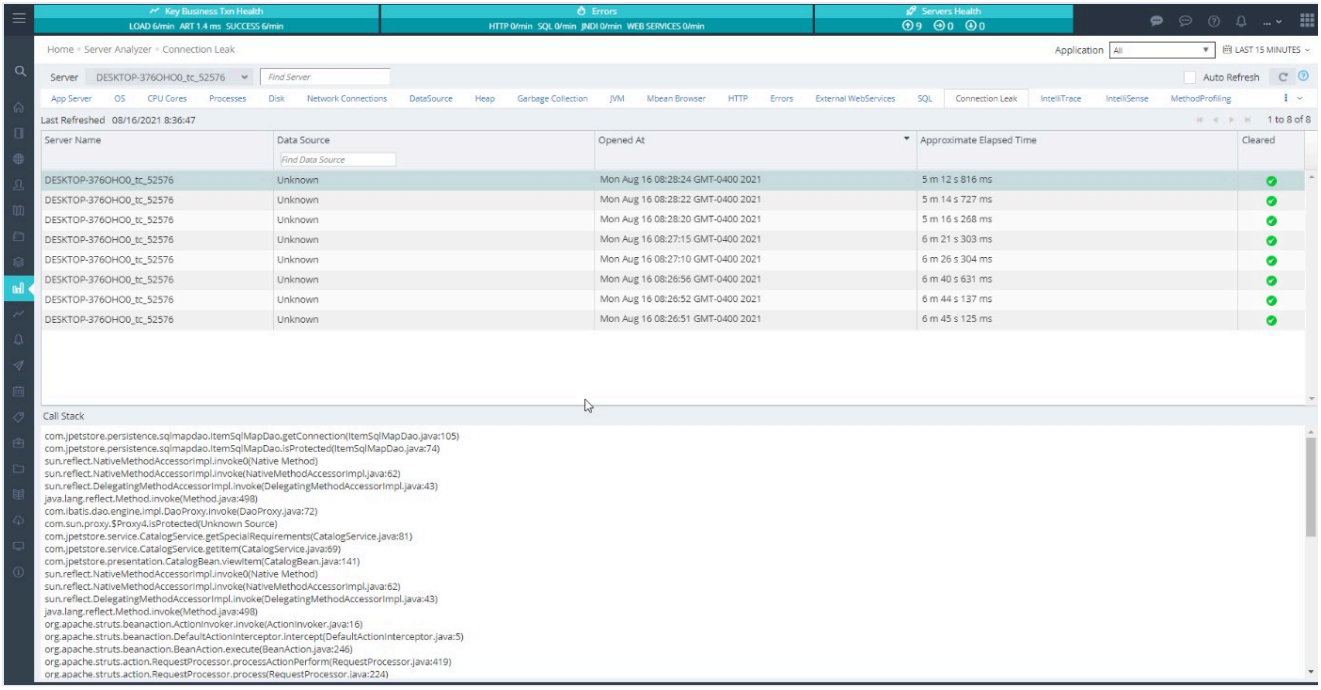



This screen highlights SQL queries vulnerable to injection attacks in red color.



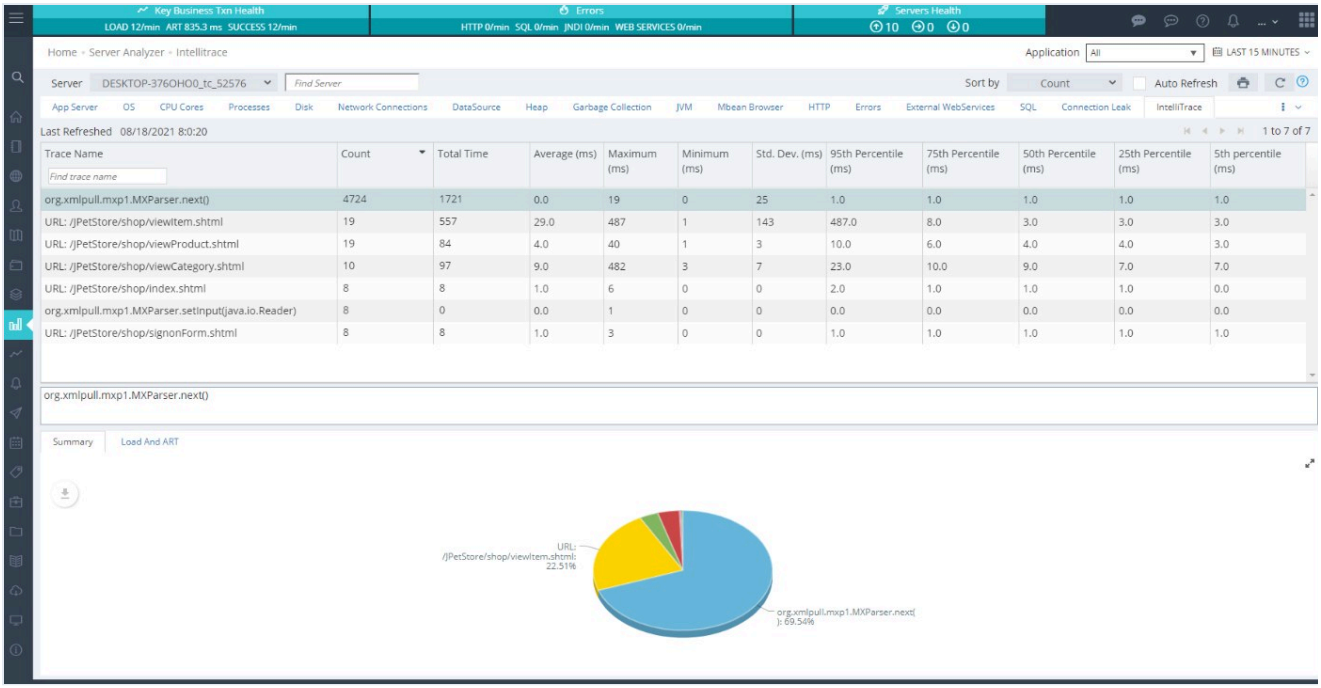

## Connection Leak

JDBC Connection Profiling feature allows you to detect leaked JDBC connections, connections held open for long periods of time and code responsible for connection leaks. You can view the connections that are currently in open state and clicking on an open connection will display the call stack which requested the connection. Statistics displayed are for the selected server in the UI.



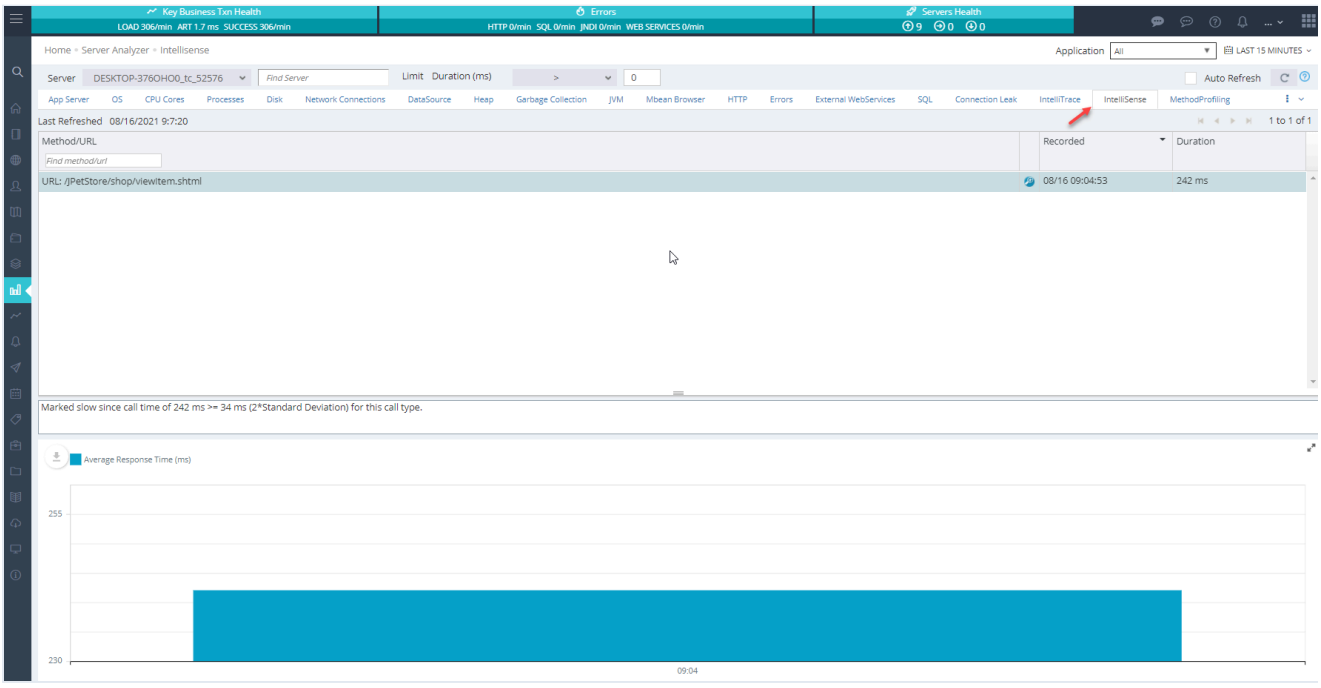

## IntelliTrace

IntelliTrace decides what methods to track dynamically and decides the depth of the call stack based on invocation counts and time spent on each individual method to minimize the overhead of tracing. IntelliTrace does what the real experts do with the profiling data - illuminate data that has little impact but it does it in real time. This way no time/resources are spent on gathering data that doesn't have meaningful impact on performance.



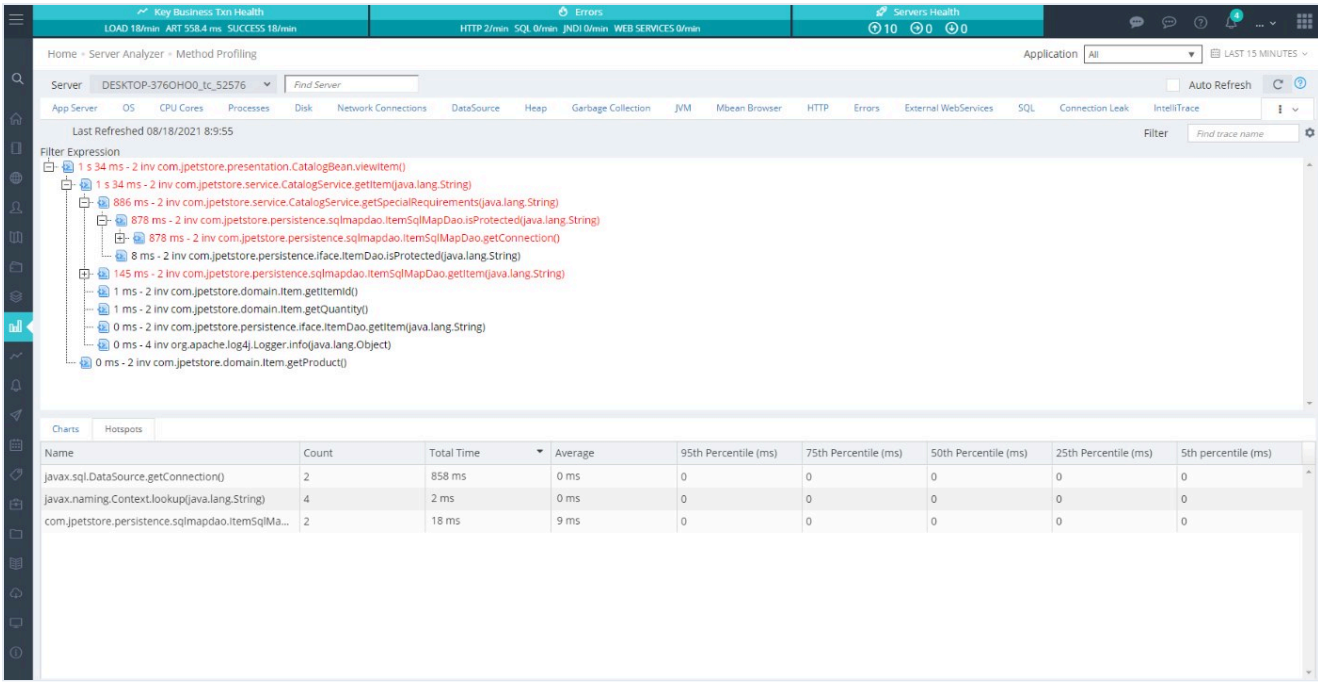

## IntelliSense

IntelliSense is a feature that works when IntelliTrace is enabled to automatically detect transaction executions that deviate considerably from the normal execution times and saves a complete call graph of the execution for later analysis.




## Method profiling

Method Profiling feature allows you to easily view the performance statistics of real-time method executions on-the-fly. This Profiling Aspect needs to be configured and deployed before you can view performance data. See Method Profiling Configuration for more information on configuration.




## JNDI

JNDI calls monitoring tracks JNDI performance and helps identify issues related to lookup delays.

## JMS

JMS calls monitoring delivers performance statistics for JMS queues and topics, including metrics such as the number of calls, total time, average duration, and various percentiles.