Using AppDynamics with LoadRunner

Exec summary

While it may seem at first look that AppDynamics is oriented towards IT Operations and DevOps, a number of our users have been using AppDynamics for years within performance engineering practices. They realize many of the same benefits as their peers in the IT Ops and DevOps practices. The key is to being proactive in avoiding performance issues prior to going live in production. The following benefits are typically associated with integration of AppDynamics with load testing technologies:

- Gaining deep application-level visibility to answer the 'Why?' vs. simply determining 'What?'
- Rapidly pinpointing bottlenecks in their environment
- Comparing results across releases and performance tests
- Improving performance by measuring the end-to-end performance from the end-user perspective
- Increasing scalability of the application by optimizing software and avoiding unnecessary hardware investments

While some of this discussion centers on LoadRunner specifically, it is important to note that these use-cases may be applied using any load testing solution and methodology, not just LoadRunner.

LoadRunner transaction integration

The simplest and fastest way to get LoadRunner transactions in AppDynamics is 'painting' your LoadRunner requests. This method involves adding a custom header that contains the LR transaction name in the request so that AppDynamics can display the transactions exactly as they appear in LoadRunner. Adding a custom header in LoadRunner is a simple process using LoadRunner's built in capability to set headers, including custom headers. The end-result gives you the same visibility as you have in LoadRunner, but with the following added benefits:

- Deep code-level visibility, in the context of a LoadRunner Transaction, end-to-end
- Application container-level data such as Heap size and usage along with standard and custom JMX metrics
- Deep database visibility, in the context of the transaction, including details SQL breakdowns
- Extended visibility into application infrastructure (CPU, Disk, Network, and Memory).

AppDynamics provides these benefits automatically without the need to configure multiple HP tools, such as HP SiteScope, and then configuring LoadRunner to pull these metrics from HP SiteScope.

It is a simple two-step process to gain this visibility:

1. A minor modification to your VUGen scripts to set the header, using the web_add_header() function.
2. A minor configuration tweak in AppDynamics to detect this new header and name transactions accordingly.

In addition, the AppDynamics integrations team is working on a utility that will help make these changes in the VUGen script automatically (as of October 2015).

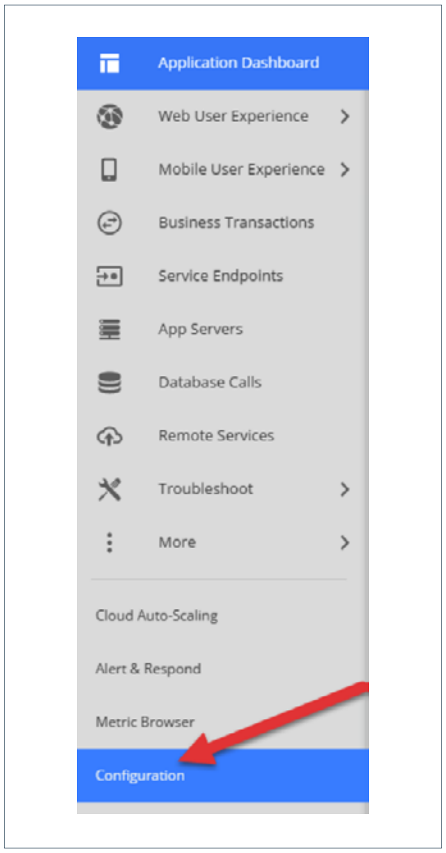
LoadRunner script changes

Here is a sample edit to a VUGen script. The highlighted lines are what would be added to an existing script allowing for AppDynamics to detect the request as a LoadRunner Transaction and name it accordingly. This process inserts a new header into the requests from LoadRunner and includes the specific LoadRrunner transaction name. We then configure AppDynamics to detect this special header and use it to automatically name the transactions discovered, avoiding manual configuration:

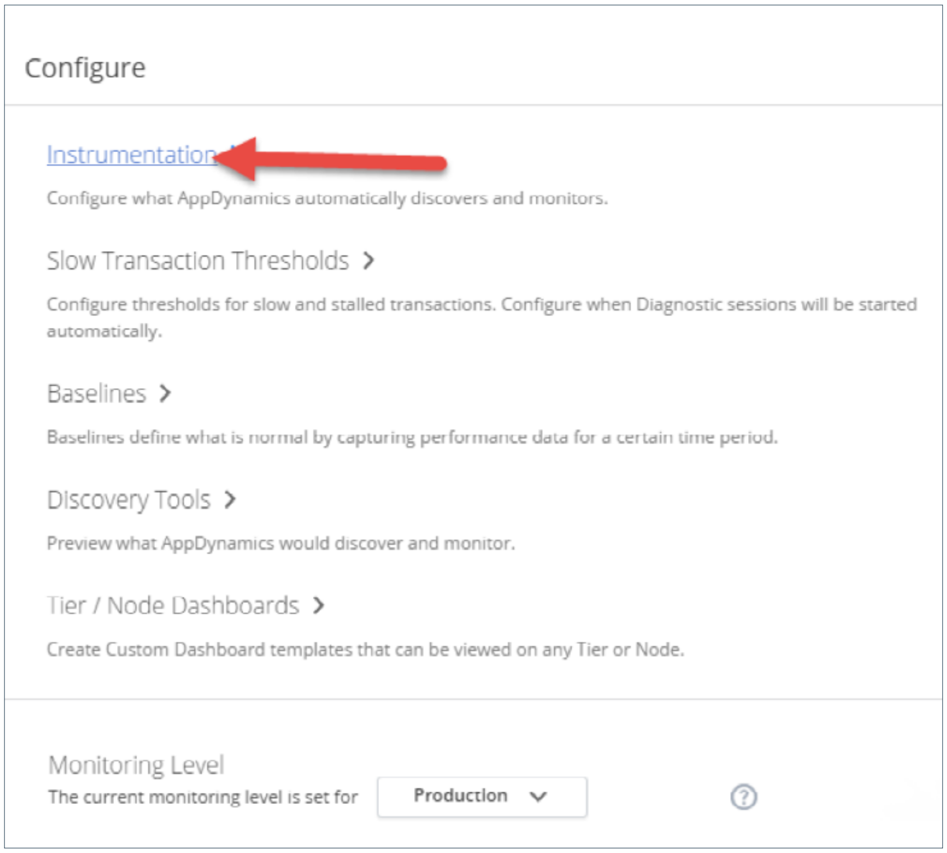
```
web_add_header(“AppD_Header”, “TxnA”);
lr_Start_transaction(“TxnA”);
    web_submit_req(“”);
lr-end-transaction(“TxnA”, LR_AUTO);
web_add_header(“AppD_Header”, “TxnB”);
lr-Start-transaction(“TxnB”);
    Web_URL(“”);
lr-end-transaction(“TxnB”, LR_AUTO);
…….
web_add_header(“AppD_Header”, “TxnZ”);
lr-Start-transaction(“TxnZ”);
    Web_custom_request(“”);
lr-end-transaction(“TxnZ”, LR_AUTO);
```

AppDynamics configuration for detecting LoadRunner transactions

Here is a sample configuration necessary on the AppDynamics side to allow for the automatic transaction naming:



Select configuration from the left-hand menu.



Configure

Instrumentation

Configure what AppDynamics automatically discovers and monitors.

Slow Transaction Thresholds

Configure thresholds for slow and stalled transactions. Configure when Diagnostic sessions will be started automatically.

Baselines

Baselines define what is normal by capturing performance data for a certain time period.

Discovery Tools

Preview what AppDynamics would discover and monitor.

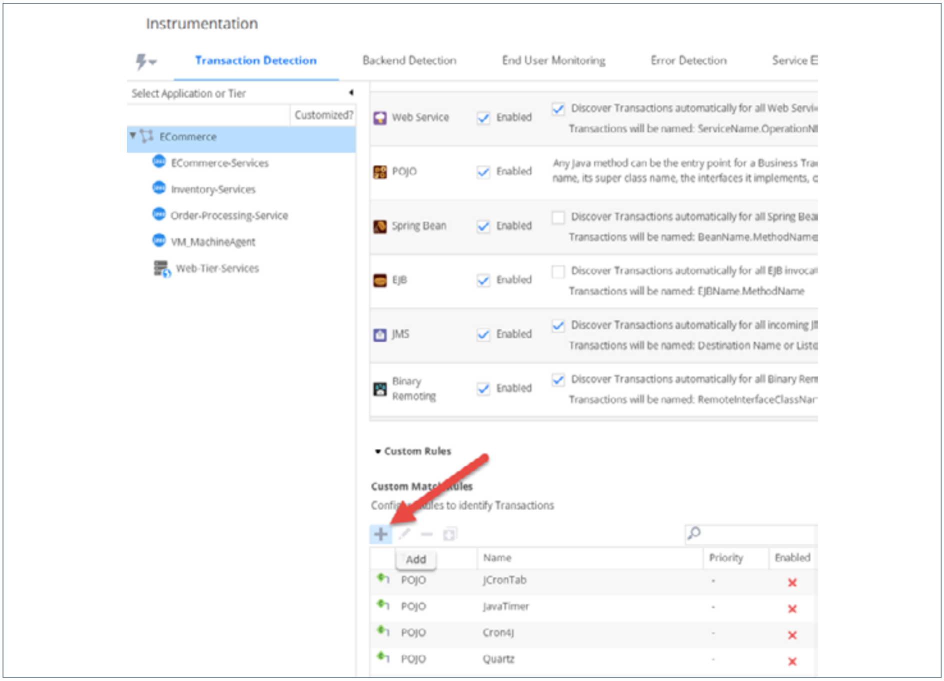
Tier / Node Dashboards

Create Custom Dashboard templates that can be viewed on any Tier or Node.

Monitoring Level

The current monitoring level is set for Production

Choose Instrumentation.



Web Service	✓ Enabled	✓ Discover Transactions automatically for all Web Servi Transactions will be named: ServiceName.OperationN
POJO	✓ Enabled	Any Java method can be the entry point for a Business Tra name, its super class name, the interfaces it implements, c
Spring Bean	✓ Enabled	☐ Discover Transactions automatically for all Spring Bea Transactions will be named: BeanName.MethodName
EJB	✓ Enabled	☐ Discover Transactions automatically for all EJB invoca Transactions will be named: EJBName.MethodName
JMS	✓ Enabled	✓ Discover Transactions automatically for all incoming J Transactions will be named: Destination Name or Listo
Binary Remoting	✓ Enabled	✓ Discover Transactions automatically for all Binary Rem Transactions will be named: RemoteInterfaceClassNar

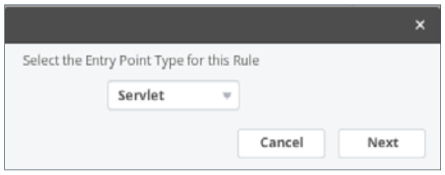
Custom Rules

Custom Match Rules

Configure rules to identify Transactions

	Name	Priority	Enabled
POJO	JCronTab	-	✗
POJO	JavaTimer	-	✗
POJO	Cron4J	-	✗
POJO	Quartz	-	✗

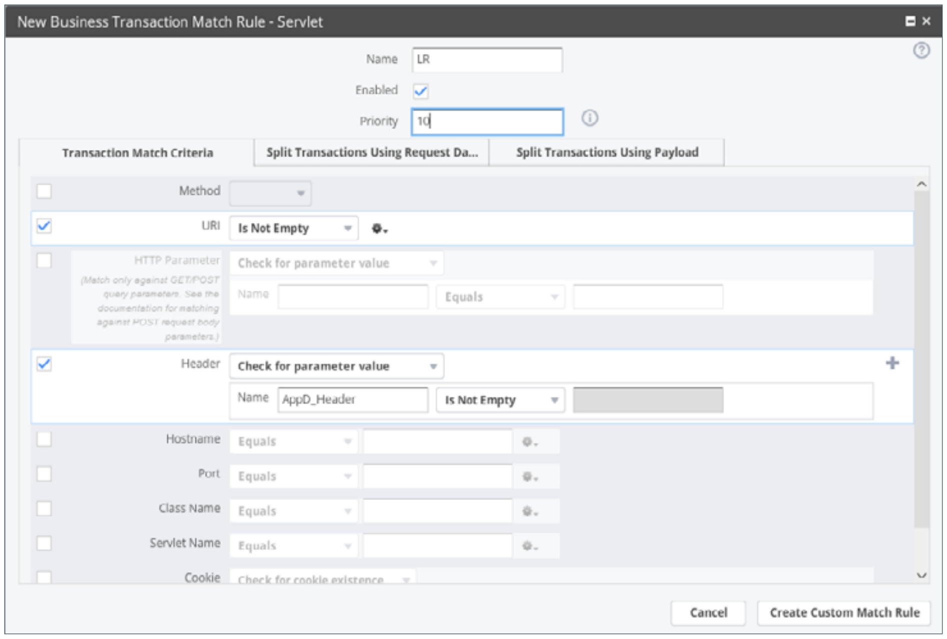
Select your appropriate application and/or tier from the menu on the left, then add a new custom match rule.



Select the Entry Point Type for this Rule

Servlet

Choose 'Servlet' as your rule type.

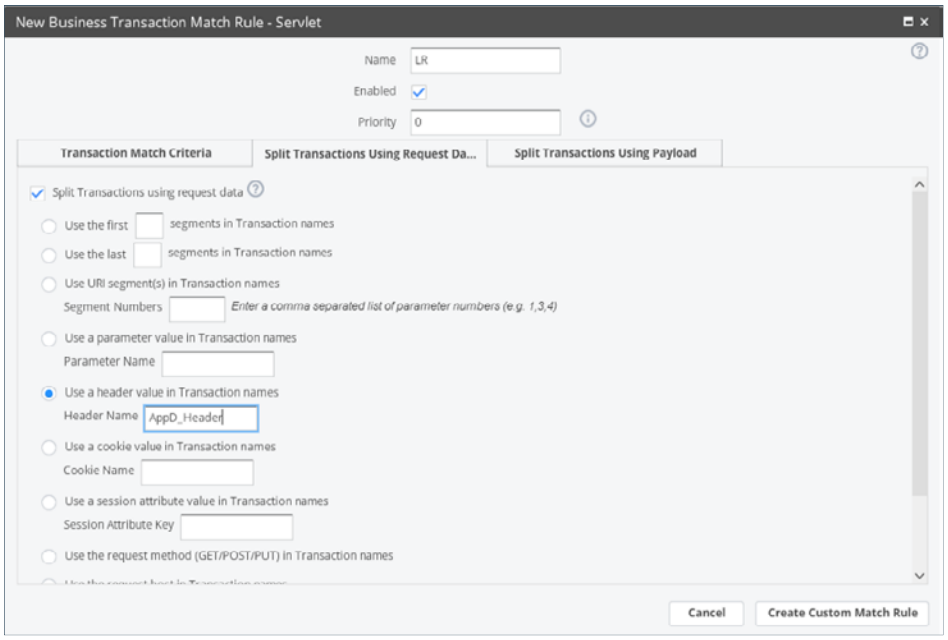

New Business Transaction Match Rule - Servlet

Name: LR
Enabled: ✓
Priority: 10

Transaction Match Criteria | Split Transactions Using Request Da... | Split Transactions Using Payload

- [ ] Method
- [✓] URI: Is Not Empty
- [ ] HTTP Parameter (Match only against GET/POST query parameters. See the documentation for matching against POST request body parameters.): Check for parameter value; Name; Equals
- [✓] Header: Check for parameter value; Name: AppD_Header; Is Not Empty
- [ ] Hostname: Equals
- [ ] Port: Equals
- [ ] Class Name: Equals
- [ ] Servlet Name: Equals
- [ ] Cookie: Check for cookie existence

Give the rule a name like "LoadRunner". On the Transaction Match Criteria tab, we are going to check for the existence of the "AppD_Header", and that it has a value. Also configure a sanity-check in the rule that the URI is not empty.


New Business Transaction Match Rule - Servlet

Name: LR
Enabled: ✓
Priority: 0

Transaction Match Criteria | Split Transactions Using Request Da... | Split Transactions Using Payload

- [✓] Split Transactions using request data
  - ○ Use the first ___ segments in Transaction names
  - ○ Use the last ___ segments in Transaction names
  - ○ Use URI segment(s) in Transaction names
    Segment Numbers ___ *Enter a comma separated list of parameter numbers (e.g. 1,3,4)*
  - ○ Use a parameter value in Transaction names
    Parameter Name ___
  - ● Use a header value in Transaction names
    Header Name: AppD_Header
  - ○ Use a cookie value in Transaction names
    Cookie Name ___
  - ○ Use a session attribute value in Transaction names
    Session Attribute Key ___
  - ○ Use the request method (GET/POST/PUT) in Transaction names
  - ○ Use the request host in Transaction names

Select the Split Transactions tab. Choose 'Use header value...' option to use the value of the header to name the transactions based on the value of the header.

This rule would only need to be created once per application in the Controller, regardless of how many LoadRunner transactions or scripts are executed. It applies only to LoadRunner traffic, hence traffic created by actual users or applications would be unaffected. This traffic would be normally detected and classified by AppDynamics. To summarize, after making the changes, AppDynamics only looks for the special header named “AppD_Header”. If this header exists, AppDynamics names the transaction based on the value the header, which is the LoadRunner transaction name. The end-result in AppDynamics will look similar to the below:

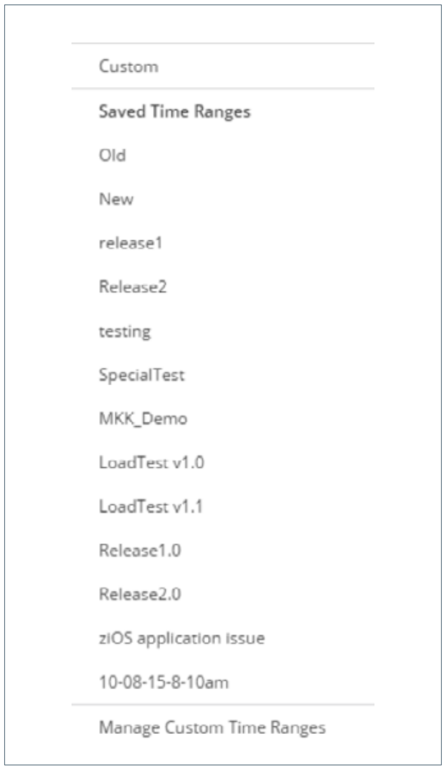
```
LR.TxnA

LR.TxnB

...

LR.TxnZ
```

Once again, this configuration applies specifically to LoadRunner generated transactions and not to other traffic (such as activity generated by QA Users) that is occurring in the system. The net results are easily-sorted and -filterable views that are specific to the load test.

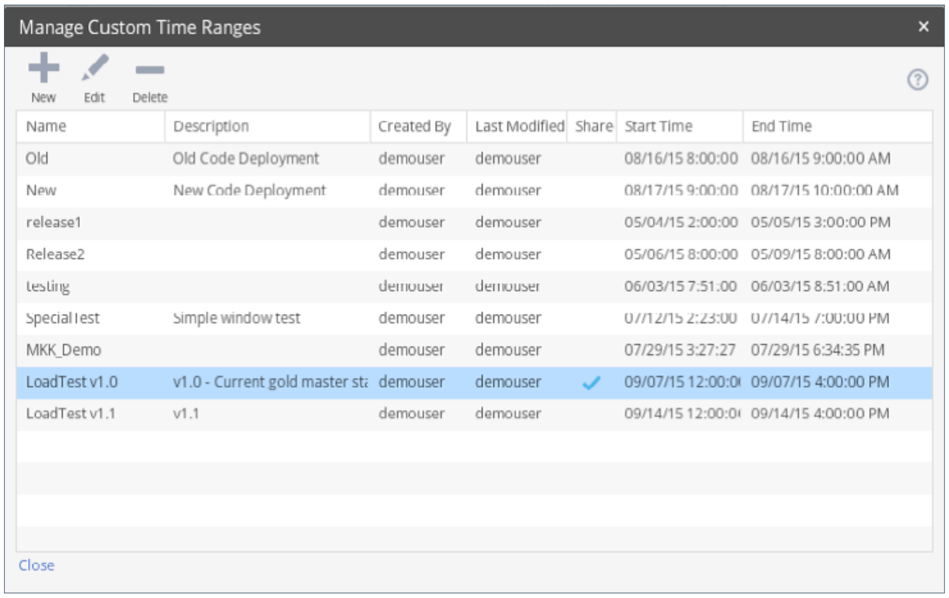
Performance engineering use-cases

Custom time ranges

Custom time ranges are simply ranges that a user can specify in AppDynamics for quick and easy access. In Production use-cases, these correspond with an application release, or even a production event such as an outage or performance degradation. For performance engineering, these typically map to timeframes which correspond to a load test or an application build.
This allows for fast access to specific load tests and time ranges when using the “Compare Releases” functionality, reviewing historical time frames, or simply focusing on data specific to a specific load test.



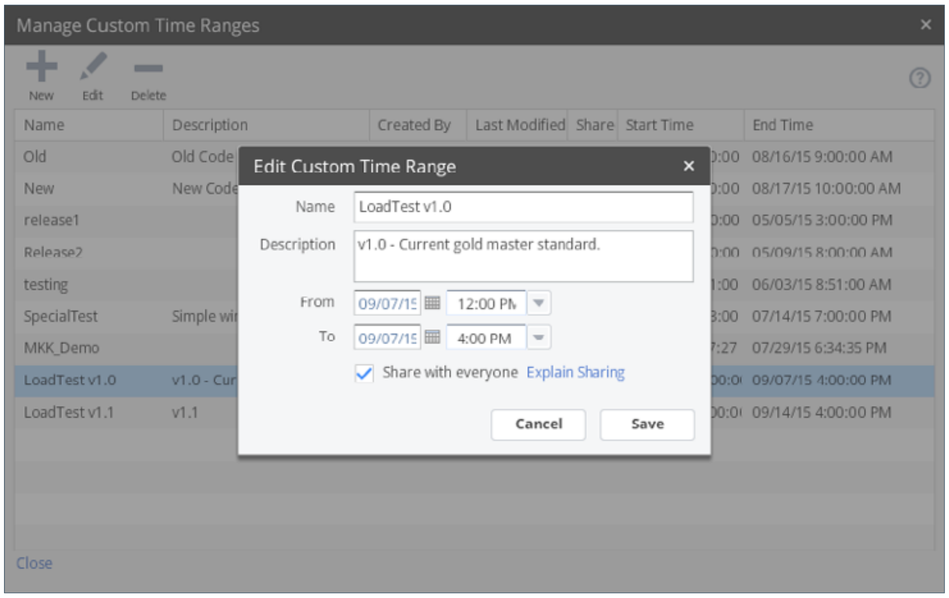
Saved Time Ranges appear within the normal time range selection box, but are located at the bottom.



Manage Custom Time Ranges

Name	Description	Created By	Last Modified	Share	Start Time	End Time
Old	Old Code Deployment	demouser	demouser		08/16/15 8:00:00	08/16/15 9:00:00 AM
New	New Code Deployment	demouser	demouser		08/17/15 9:00:00	08/17/15 10:00:00 AM
release1		demouser	demouser		05/04/15 2:00:00	05/05/15 3:00:00 PM
Release2		demouser	demouser		05/06/15 8:00:00	05/09/15 8:00:00 AM
testing		demouser	demouser		06/03/15 7:51:00	06/03/15 8:51:00 AM
SpecialTest	Simple window test	demouser	demouser		07/12/15 2:23:00	07/14/15 7:00:00 PM
MKK_Demo		demouser	demouser		07/29/15 3:27:27	07/29/15 6:34:35 PM
LoadTest v1.0	v1.0 - Current gold master st	demouser	demouser	✓	09/07/15 12:00:0	09/07/15 4:00:00 PM
LoadTest v1.1	v1.1	demouser	demouser		09/14/15 12:00:0	09/14/15 4:00:00 PM

Selecting the Manage Custom Time Ranges option allows for management and definition of custom time ranges.



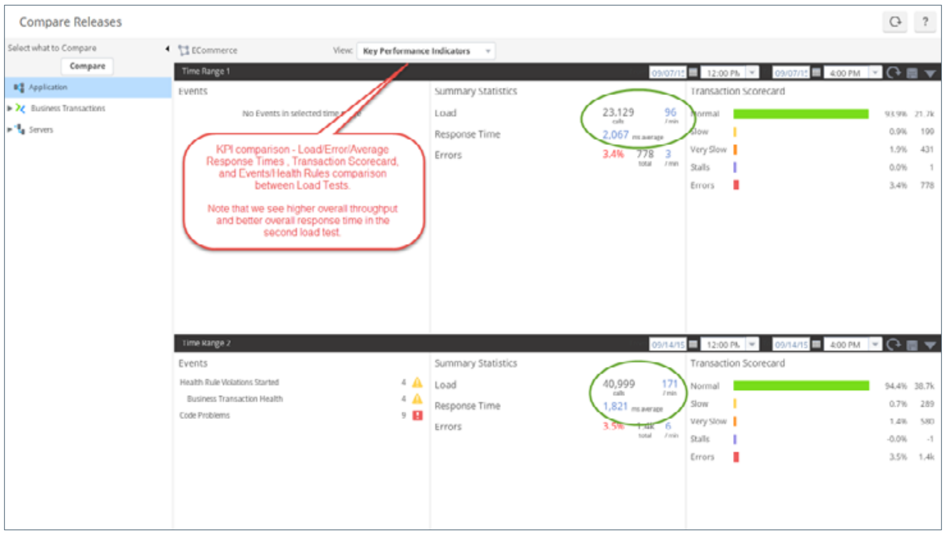
Edit Custom Time Range

Name	LoadTest v1.0
Description	v1.0 - Current gold master standard.
From	09/07/15 12:00 PM
To	09/07/15 4:00 PM
✓	Share with everyone

Defining a time range is simple – choose the times that your load test ran, and provide a description.

Compare releases

The Compare Releases functionality allows for comparing performance of the application, inclusive of various KPIs, and trending metrics between any two timeframes. This comparison can be made at the application-level or specific to individual elements. Use cases include focusing on a single server or individual transaction to allow for performance analysis and tuning. For performance engineers, being able to quickly compare results between two load tests, and identify deltas leads to rapid root-cause analysis, saving time and effort in every load testing cycle.



Compare Releases

Select what to Compare | Compare | Application | Business Transactions | Servers

ECommerce — View: Key Performance Indicators

Time Range 1 — 09/07/15 12:00 PM – 09/07/15 4:00 PM

Events

No Events in selected time range

Summary Statistics

Load	23,129 calls		96 /min
Response Time	2,067 ms average		
Errors	3.4%	778 total	3 /min

Transaction Scorecard

Normal	93.9%	21.7k
Slow	0.9%	199
Very Slow	1.9%	431
Stalls	0.0%	1
Errors	3.4%	778

Time Range 2 — 09/14/15 12:00 PM – 09/14/15 4:00 PM

Events

Health Rule Violations Started	4
Business Transaction Health	4
Code Problems	9

Summary Statistics

Load	40,999 calls		171 /min
Response Time	1,821 ms average		
Errors	3.5%	1.4k total	6 /min

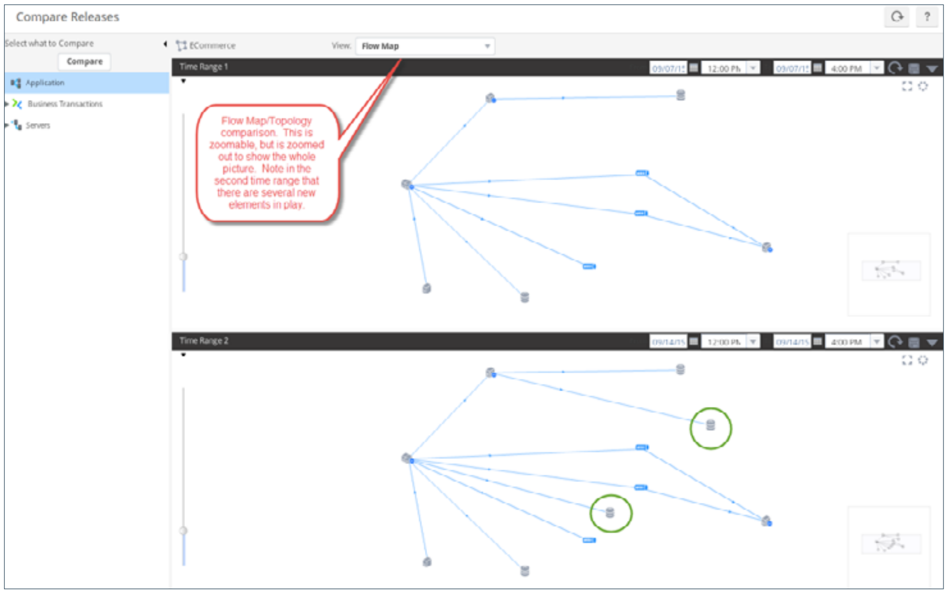
Transaction Scorecard

Normal	94.4%	38.7k
Slow	0.7%	289
Very Slow	1.4%	580
Stalls	-0.0%	-1
Errors	3.5%	1.4k

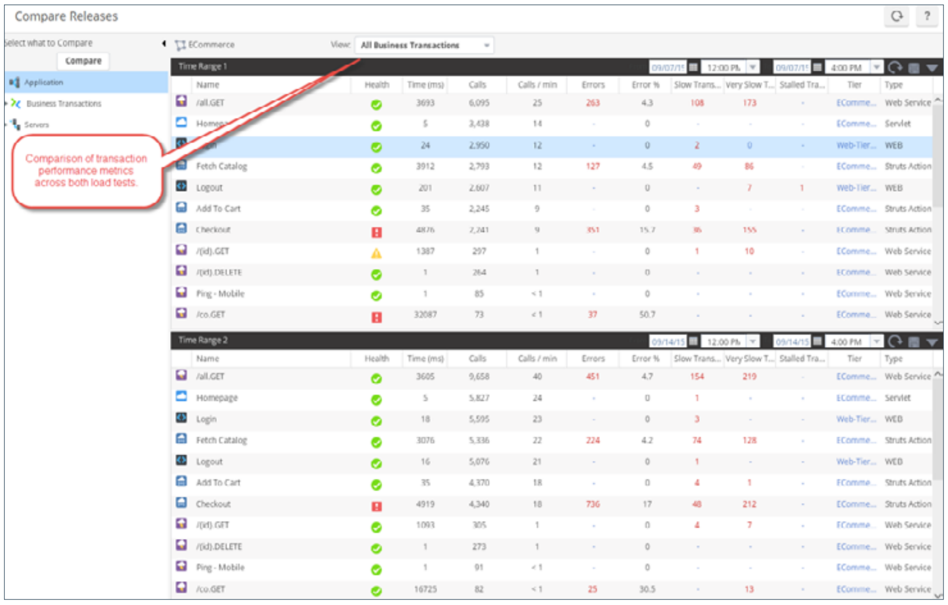
KPI comparison - Load/Error/Average Response Times , Transaction Scorecard, and Events/Health Rules comparison between Load Tests.

Note that we see higher overall throughput and better overall response time in the second load test.

The KPI view shows the top stats about the test, such as volume, average performance, the transaction scorecard and any events which occurred.


Compare Releases
Select what to Compare
Compare
Application
Business Transactions
Servers
ECommerce
View: Flow Map
Time Range 1
09/07/15 12:00 PM 09/07/15 4:00 PM
Flow Map/Topology comparison. This is zoomable, but is zoomed out to show the whole picture. Note in the second time range that there are several new elements in play.
Time Range 2
09/14/15 12:00 PM 09/14/15 4:00 PM

The Flow Map view compares application data flow and topology. This identifies if the application data flow has changed and quickly allows for identification of new or missing elements in the transaction path. This can be done at the individual transaction level or at the aggregate application level. Note in the example above, the two green circled elements do not show up in the first time range.


Compare Releases
Select what to Compare
Compare
Application
Business Transactions
Servers
ECommerce
View: All Business Transactions
Comparison of transaction performance metrics across both load tests.

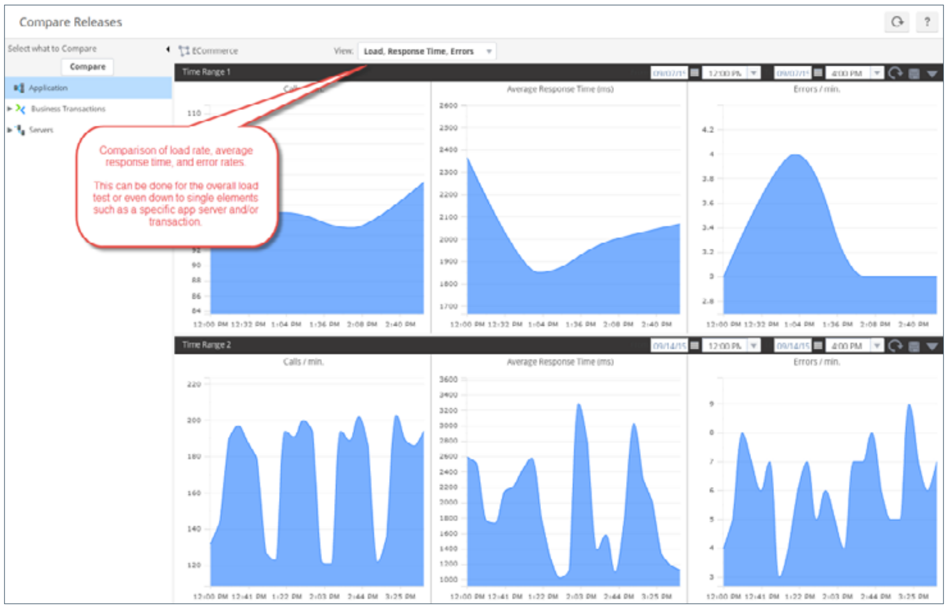
Time Range 1 — 09/07/15 12:00 PM – 09/07/15 4:00 PM

Name	Health	Time (ms)	Calls	Calls / min	Errors	Error %	Slow Trans...	Very Slow T...	Stalled Tra...	Tier	Type
/all.GET	✓	3693	6,095	25	263	4.3	108	173	-	EComme...	Web Service
Homepage	✓	5	3,438	14	-	0	-	-	-	EComme...	Servlet
Login	✓	24	2,950	12	-	0	2	0	-	Web-Tier...	WEB
Fetch Catalog	✓	3912	2,793	12	127	4.5	49	86	-	EComme...	Struts Action
Logout	✓	201	2,607	11	-	0	-	7	1	Web-Tier...	WEB
Add To Cart	✓	35	2,245	9	-	0	3	-	-	EComme...	Struts Action
Checkout	!	4876	2,241	9	351	15.7	36	155	-	EComme...	Struts Action
/{id}.GET	⚠	1387	297	1	-	0	1	10	-	EComme...	Web Service
/{id}.DELETE	✓	1	264	1	-	0	-	-	-	EComme...	Web Service
Ping - Mobile	✓	1	85	< 1	-	0	-	-	-	EComme...	Web Service
/co.GET	!	32087	73	< 1	37	50.7	-	-	-	EComme...	Web Service

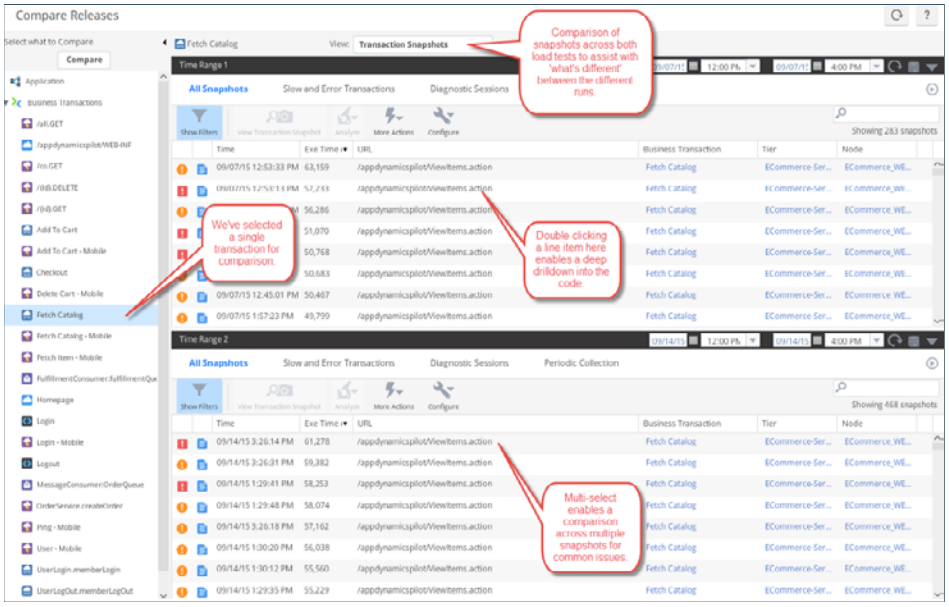
Time Range 2 — 09/14/15 12:00 PM – 09/14/15 4:00 PM

Name	Health	Time (ms)	Calls	Calls / min	Errors	Error %	Slow Trans...	Very Slow T...	Stalled Tra...	Tier	Type
/all.GET	✓	3605	9,658	40	451	4.7	154	219	-	EComme...	Web Service
Homepage	✓	5	5,827	24	-	0	1	-	-	EComme...	Servlet
Login	✓	18	5,595	23	-	0	3	-	-	Web-Tier...	WEB
Fetch Catalog	✓	3076	5,336	22	224	4.2	74	128	-	EComme...	Struts Action
Logout	✓	16	5,076	21	-	0	1	-	-	Web-Tier...	WEB
Add To Cart	✓	35	4,370	18	-	0	4	1	-	EComme...	Struts Action
Checkout	!	4919	4,340	18	736	17	48	212	-	EComme...	Struts Action
/{id}.GET	✓	1093	305	1	-	0	4	7	-	EComme...	Web Service
/{id}.DELETE	✓	1	273	1	-	0	-	-	-	EComme...	Web Service
Ping - Mobile	✓	1	91	< 1	-	0	-	-	-	EComme...	Web Service
/co.GET	✓	16725	82	< 1	25	30.5	-	13	-	EComme...	Web Service

The All Business Transactions View provides a data grid with a detailed status of each individual transaction in the load test.



The Load, Response Time, Errors View provides a visual comparison of the trends over time for the load test.


Compare Releases
Select what to Compare
Compare
Application
Business Transactions
/all.GET
/appdynamicspilot/WEB-INF
/cs.GET
/0d).DELETE
/0d).GET
Add To Cart
Add To Cart - Mobile
Checkout
Delete Cart - Mobile
Fetch Catalog
Fetch Catalog - Mobile
Fetch Item - Mobile
FulfillmentConsumer.fulfillmentQue
Homepage
Login
Login - Mobile
Logout
MessageConsumer.OrderQueue
OrderService.createOrder
Ping - Mobile
User - Mobile
UserLogin.memberLogin
UserLogOut.memberLogOut
Fetch Catalog
View: Transaction Snapshots
Comparison of snapshots across both load tests to assist with 'what's different' between the different runs.
We've selected a single transaction for comparison.
Double clicking a line item here enables a deep drilldown into the code.
Multi-select enables a comparison across multiple snapshots for common issues.
Time Range 1
09/07/15 12:00 PM 09/07/15 4:00 PM
All Snapshots
Slow and Error Transactions
Diagnostic Sessions
Show Filters
View Transaction Snapshot
Analyze
More Actions
Configure
Showing 283 snapshots
Time Range 2
09/14/15 12:00 PM 09/14/15 4:00 PM
Periodic Collection
Showing 468 snapshots

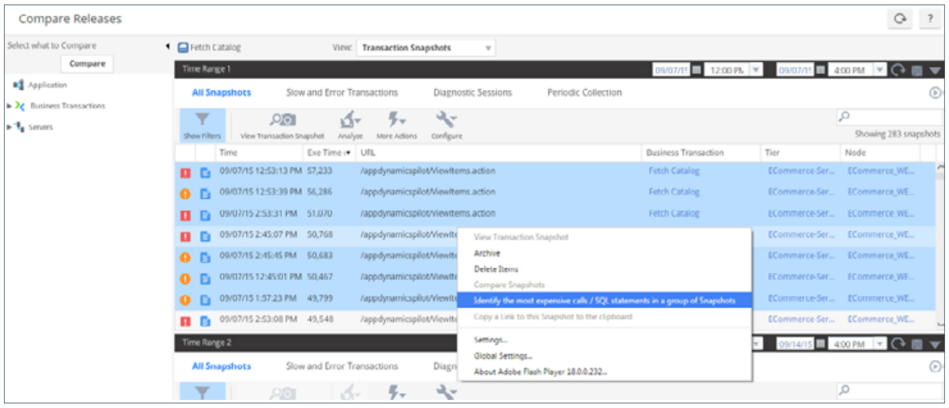
Time Range 1

Time	Exe Time	URL	Business Transaction	Tier	Node
09/07/15 12:53:33 PM	63,159	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 12:53:13 PM	57,233	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
M	56,286	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
	51,070	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
	50,768	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
	50,683	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 12:45:01 PM	50,467	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 1:57:23 PM	49,799	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...

Time Range 2

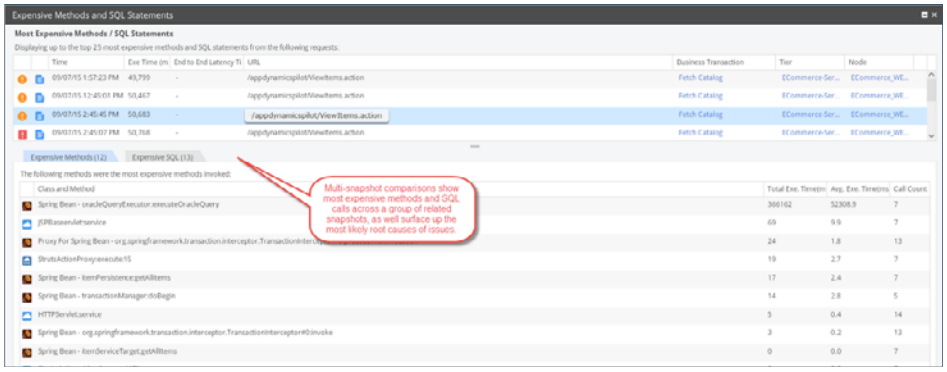
Time	Exe Time	URL	Business Transaction	Tier	Node
09/14/15 3:26:14 PM	61,278	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/14/15 3:26:31 PM	59,382	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/14/15 1:29:41 PM	58,253	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/14/15 1:29:48 PM	58,074	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/14/15 3:26:18 PM	57,162	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/14/15 1:30:20 PM	56,038	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/14/15 1:30:12 PM	55,560	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/14/15 1:29:35 PM	55,229	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...

Transaction snapshot comparisons allow for direct comparison of transaction snapshots related to specific transaction between both load tests. For reference, a snapshot is a deep-level capture of a single occurrence of a transaction that allows one to see code-level details of that transaction's execution. By comparing snapshots, users can identify differences in code execution paths and identify slowness down to the individually-executed methods.



Time	Exe Time	URL	Business Transaction	Tier	Node
09/07/15 12:53:13 PM	57,233	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 12:53:39 PM	56,286	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 2:53:31 PM	51,070	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 2:45:07 PM	50,768	/appdynamicspilot/ViewIt		ECommerce-Ser...	ECommerce_WE...
09/07/15 2:45:45 PM	50,683	/appdynamicspilot/ViewIt		ECommerce-Ser...	ECommerce_WE...
09/07/15 12:45:01 PM	50,467	/appdynamicspilot/ViewIt		ECommerce-Ser...	ECommerce_WE...
09/07/15 1:57:23 PM	49,799	/appdynamicspilot/ViewIt		ECommerce-Ser...	ECommerce_WE...
09/07/15 2:53:08 PM	49,548	/appdynamicspilot/ViewIt		ECommerce-Ser...	ECommerce_WE...

Multiple snapshots may also be compared to quickly highlight 'hotspots' in code or to see a trend in long running methods or SQL calls.


Expensive Methods and SQL Statements
Expensive Methods (12)
Expensive SQL (13)
Multi-snapshot comparisons show most expensive methods and SQL calls across a group of related snapshots, as well surface up the most likely root causes of issues.

Most Expensive Methods / SQL Statements

Displaying up to the top 25 most expensive methods and SQL statements from the following requests:

Time	Exe Time (m	End to End Latency Ti	URL	Business Transaction	Tier	Node
09/07/15 1:57:23 PM	49,799	-	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 12:45:01 PM	50,467	-	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 2:45:45 PM	50,683	-	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...
09/07/15 2:45:07 PM	50,768	-	/appdynamicspilot/ViewItems.action	Fetch Catalog	ECommerce-Ser...	ECommerce_WE...

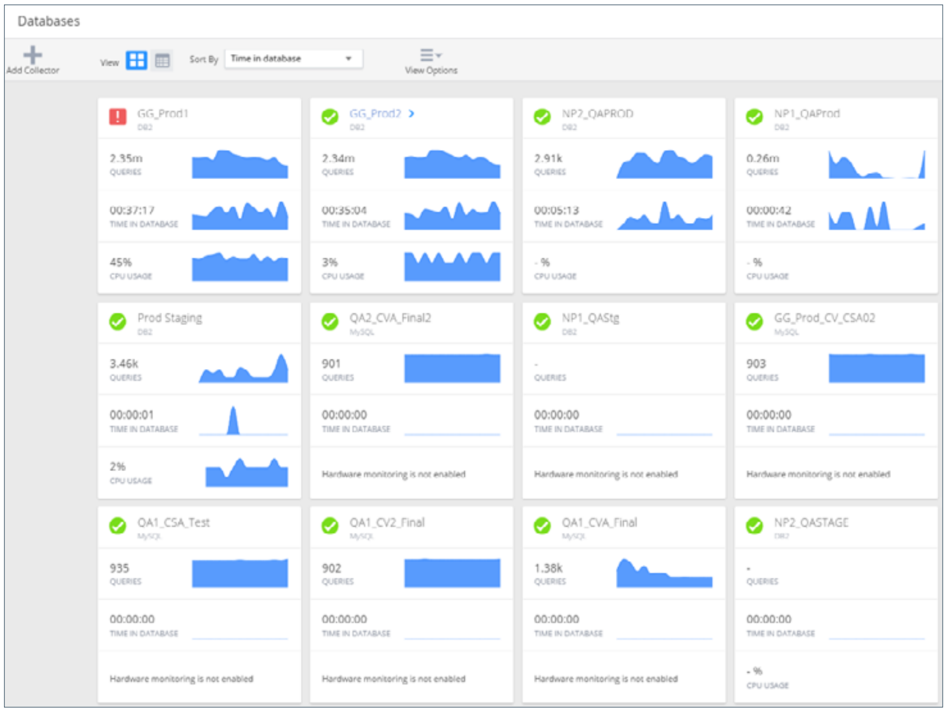
The following methods were the most expensive methods invoked:

Class and Method	Total Exe. Time(m	Avg. Exe. Time(ms	Call Count
Spring Bean - oracleQueryExecutor.executeOracleQuery	366162	52308.9	7
JSPBaseervlet:service	69	9.9	7
Proxy For Spring Bean - org.springframework.transaction.interceptor.TransactionInterc	24	1.8	13
StrutsActionProxy:execute:15	19	2.7	7
Spring Bean - itemPersistence.getAllItems	17	2.4	7
Spring Bean - transactionManager.doBegin	14	2.8	5
HTTPServlet.service	5	0.4	14
Spring Bean - org.springframework.transaction.interceptor.TransactionInterceptor#0.invoke	3	0.2	13
Spring Bean - itemServiceTarget.getAllItems	0	0.0	7

Detailed view of a multi-snapshot comparison, highlighting top most expensive methods and SQL calls.

Database visibility

AppDynamics further extends application visibility from the code-level into the database, providing deep-level visibility in the SQL execution and detailed database performance metrics. This functionality aids performance engineers by providing a contexualized-drilldown from the application into the database, acting as a bridge between the application team and the DBAs. It provides a common viewpoint while showing data in context that is relevant for both parties, and allows users to further breakdown a SQL request into where the time is being spent from a database perspective.



GG_Prod1 (DB2): 2.35m QUERIES; 00:37:17 TIME IN DATABASE; 45% CPU USAGE

GG_Prod2 (DB2): 2.34m QUERIES; 00:35:04 TIME IN DATABASE; 3% CPU USAGE

NP2_QAPROD (DB2): 2.91k QUERIES; 00:05:13 TIME IN DATABASE; - % CPU USAGE

NP1_QAProd (DB2): 0.26m QUERIES; 00:00:42 TIME IN DATABASE; - % CPU USAGE

Prod Staging (DB2): 3.46k QUERIES; 00:00:01 TIME IN DATABASE; 2% CPU USAGE

QA2_CVA_Final2 (MySQL): 901 QUERIES; 00:00:00 TIME IN DATABASE; Hardware monitoring is not enabled

NP1_QAStg (DB2): - QUERIES; 00:00:00 TIME IN DATABASE; Hardware monitoring is not enabled

GG_Prod_CV_CSA02 (MySQL): 903 QUERIES; 00:00:00 TIME IN DATABASE; Hardware monitoring is not enabled

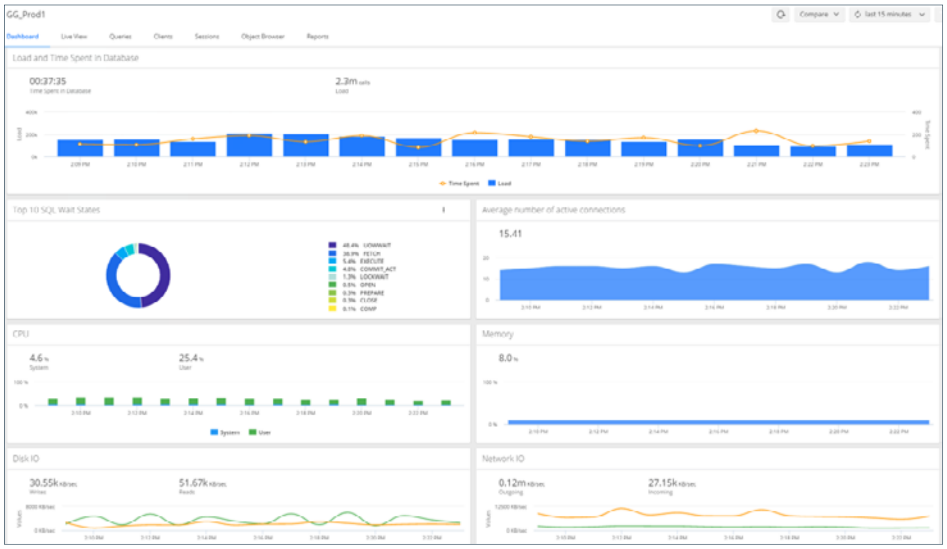
QA1_CSA_Test (MySQL): 935 QUERIES; 00:00:00 TIME IN DATABASE; Hardware monitoring is not enabled

QA1_CV2_Final (MySQL): 902 QUERIES; 00:00:00 TIME IN DATABASE; Hardware monitoring is not enabled

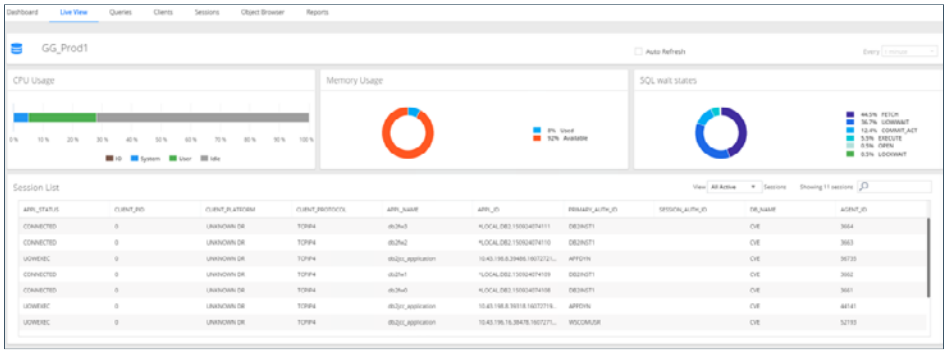
QA1_CVA_Final (MySQL): 1.38k QUERIES; 00:00:00 TIME IN DATABASE; Hardware monitoring is not enabled

NP2_QASTAGE (DB2): - QUERIES; 00:00:00 TIME IN DATABASE; - % CPU USAGE

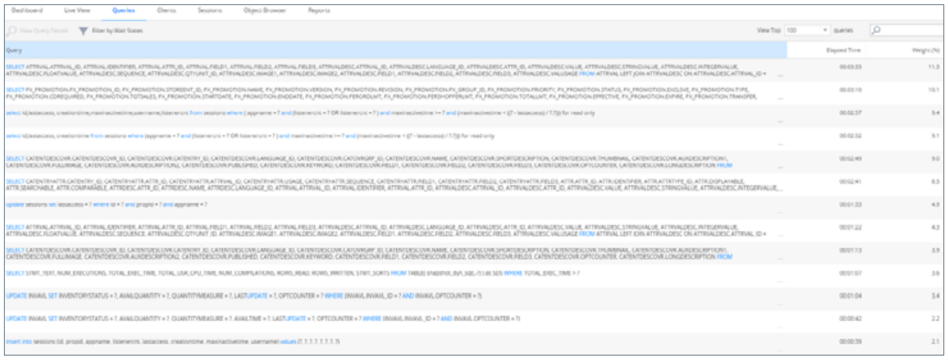
Dashboard of all of current databases and their current, live status.



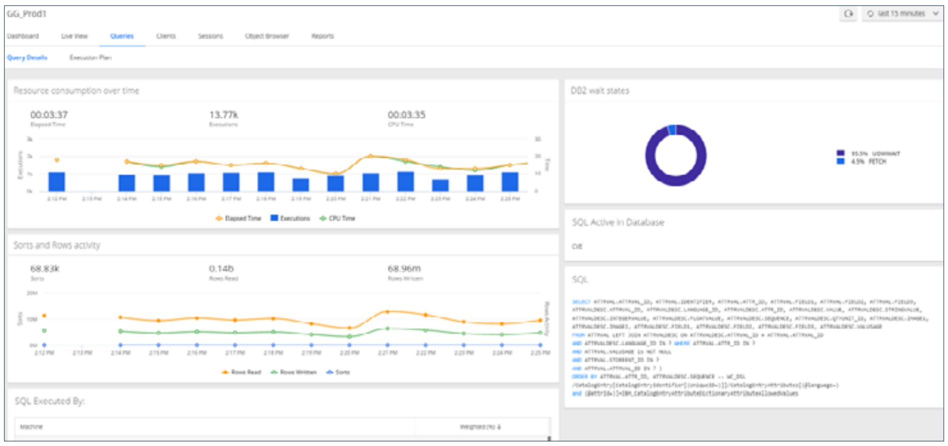
Drilldown view into a single database, with summarized view of performance, health, and activity.



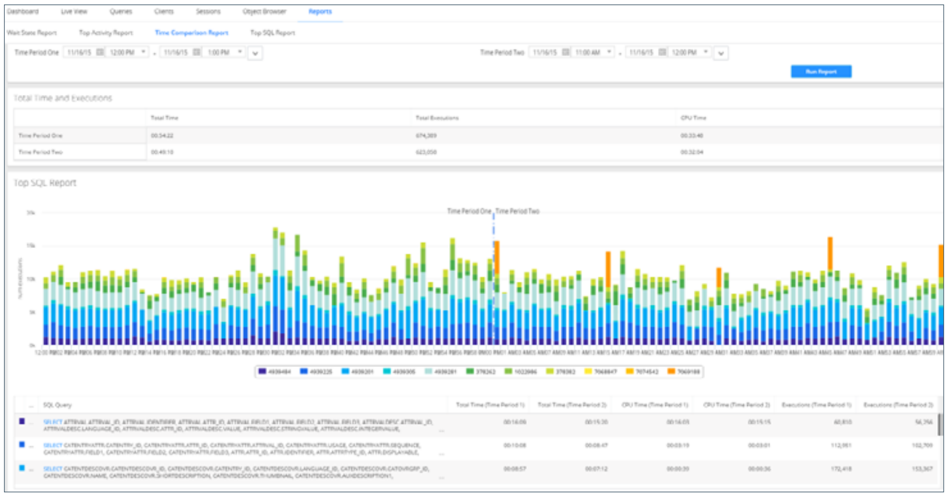
Detailed DB performance view, including wait states.



Detailed SQL view, highlighting all SQL operations, stack-ranked based on most resources consumed.



Detailed breakdown of a single SQL statement, its execution over time, clients making the request, and the ability to pull an explain plan. This enables a DBA to rapidly pinpoint the root cause for a poorly-performing SQL operation.



Time Period One 11/16/15 12:00 PM - 11/16/15 1:00 PM

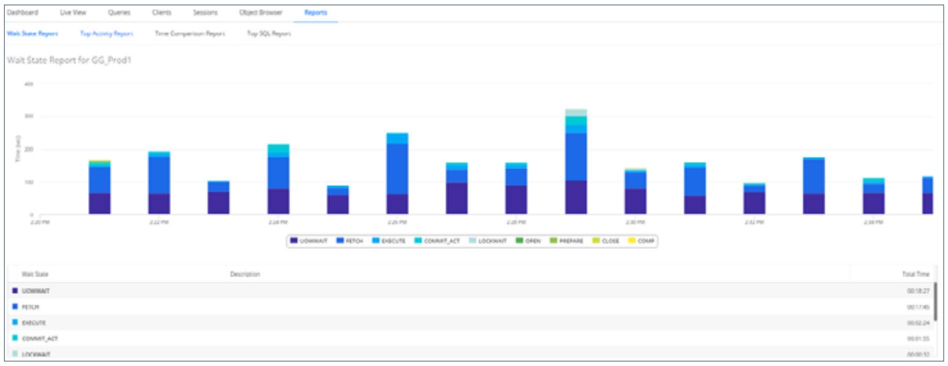
Time Period Two 11/16/15 11:00 AM - 11/16/15 12:00 PM

Total Time and Executions

	Total Time	Total Executions	CPU Time
Time Period One	00:54:22	674,309	00:33:40
Time Period Two	00:49:10	623,050	00:32:04

...	SQL Query	Total Time (Time Period 1)	Total Time (Time Period 2)	CPU Time (Time Period 1)	CPU Time (Time Period 2)	Executions (Time Period 1)	Executions (Time Period 2)
...	SELECT ATTRVAL.ATTRVAL_ID, ATTRVAL.IDENTIFIER, ATTRVAL.ATTR_ID, ATTRVAL.FIELD1, ATTRVAL.FIELD2, ATTRVAL.FIELD3, ATTRVALDESC.ATTRVAL_ID, ATTRVALDESC.LANGUAGE_ID, ATTRVALDESC.ATTR_ID, ATTRVALDESC.VALUE, ATTRVALDESC.STRINGVALUE, ATTRVALDESC.INTEGERVALUE, ...	00:16:09	00:15:20	00:16:03	00:15:15	60,810	56,256
...	SELECT CATENTRYATTR.CATENTRY_ID, CATENTRYATTR.ATTR_ID, CATENTRYATTR.ATTRVAL_ID, CATENTRYATTR.USAGE, CATENTRYATTR.SEQUENCE, CATENTRYATTR.FIELD1, CATENTRYATTR.FIELD2, CATENTRYATTR.FIELD3, ATTR.ATTR_ID, ATTR.IDENTIFIER, ATTR.ATTRTYPE_ID, ATTR.DISPLAYABLE, ...	00:10:08	00:08:47	00:03:19	00:03:01	112,951	102,709
...	SELECT CATENTDESCOVR.CATENTDESCOVR_ID, CATENTDESCOVR.CATENTRY_ID, CATENTDESCOVR.LANGUAGE_ID, CATENTDESCOVR.CATOVRGRP_ID, CATENTDESCOVR.NAME, CATENTDESCOVR.SHORTDESCRIPTION, CATENTDESCOVR.THUMBNAIL, CATENTDESCOVR.AUXDESCRIPTION1, ...	00:08:57	00:07:12	00:00:39	00:00:36	172,418	153,367

Database Time Range Comparison enables comparison of the performance of the database across two different time frames or the performance of two difference databases, such as Perf and Prod, for a similar timeframe. This provides a similar functionality for databases as Compare Releases does for deep application visibility.

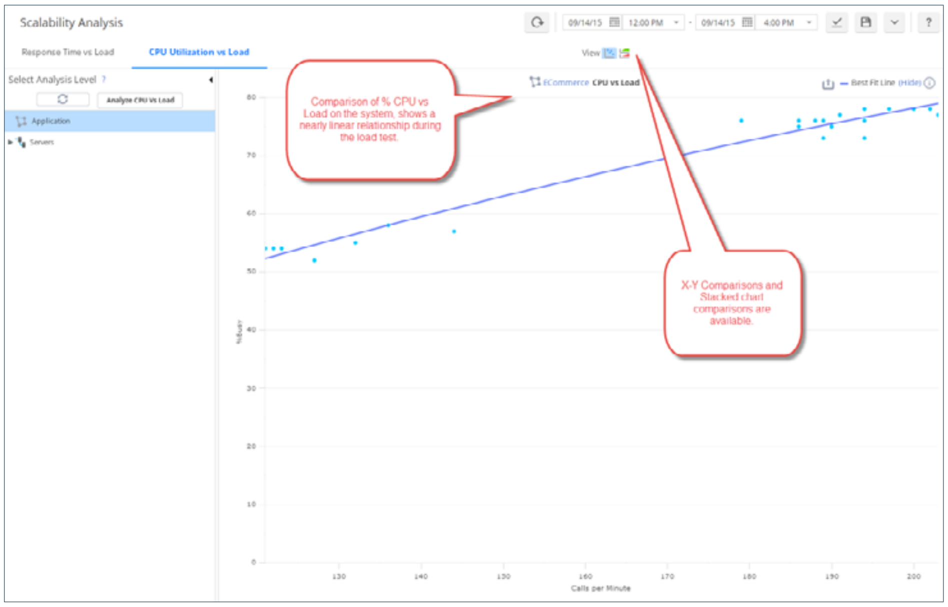


Wait State	Description	Total Time
UOWWAIT		00:18:27
FETCH		00:17:45
EXECUTE		00:02:24
COMMIT_ACT		00:01:55
LOCKWAIT		00:00:32

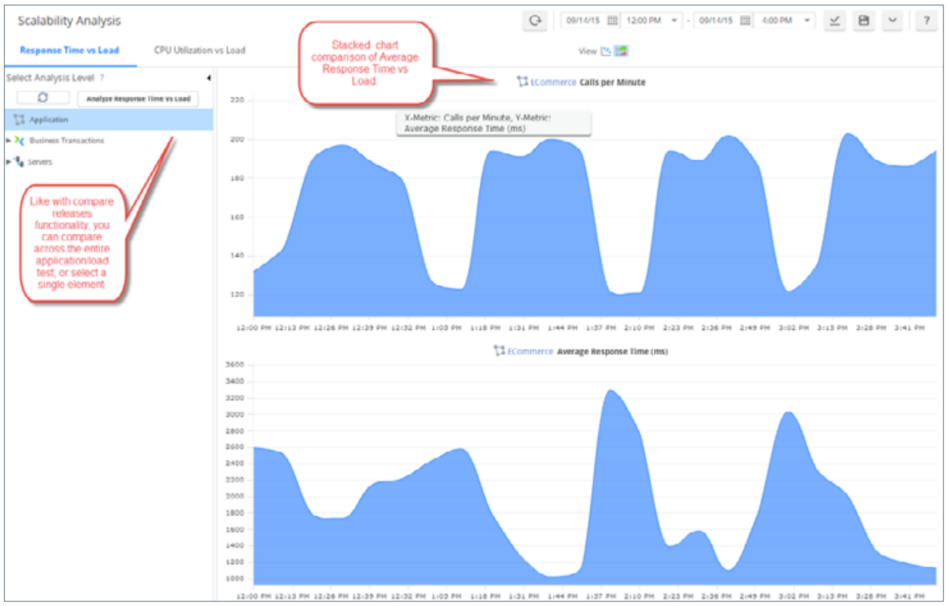
The Database Wait State Report enables detailed breakdown of the various wait states during SQL Operations for a given timeframe, enabling rapid visibility into issues such as deadlocking.

Scalability and correlation analysis

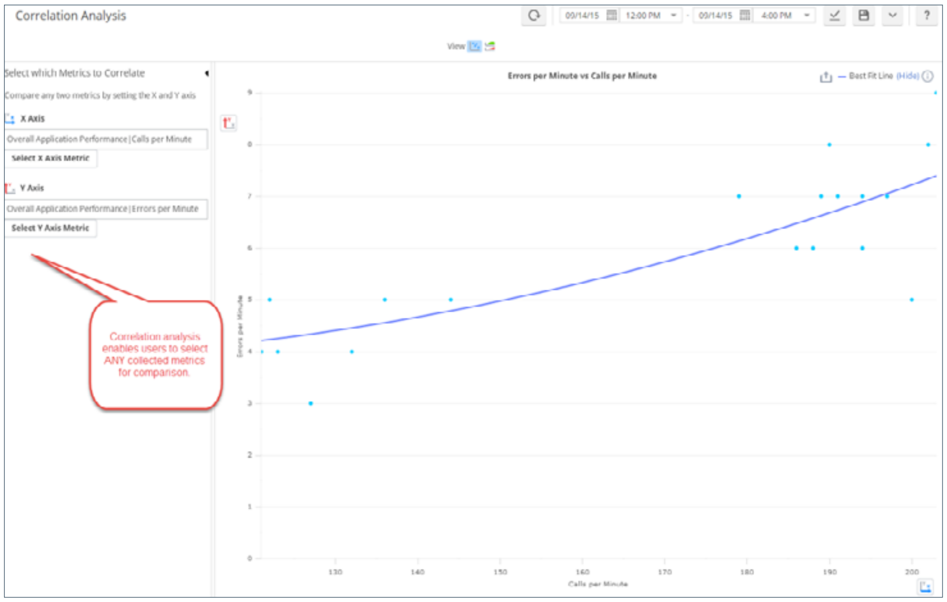
The Scalability and Correlation Analysis functions enables users to performance X-Y graphing comparisons and stacked graph comparisons of key information. This is useful when analyzing during a live load test or after a load test has completed to identify related behavioral patterns, such as inverse or direct relationships, across one or more different measurements, such as CPU vs Load or Response Time.



Scalability Analysis comes with several pre-built views for specific comparative analysis. CPU vs Load comparisons may be made at the overall application level, or against individual elements such as a single application instance or transaction. Here we are using an X-Y graph to plot the data points against each other to see a direct relationship.


Scalability Analysis
Response Time vs Load
CPU Utilization vs Load
Select Analysis Level
Analyze Response Time vs Load
Application
Business Transactions
Servers
ECommerce Calls per Minute
X-Metric: Calls per Minute, Y-Metric: Average Response Time (ms)
ECommerce Average Response Time (ms)
Stacked chart comparison of Average Response Time vs Load.
Like with compare releases functionality, you can compare across the entire application/load test, or select a single element.

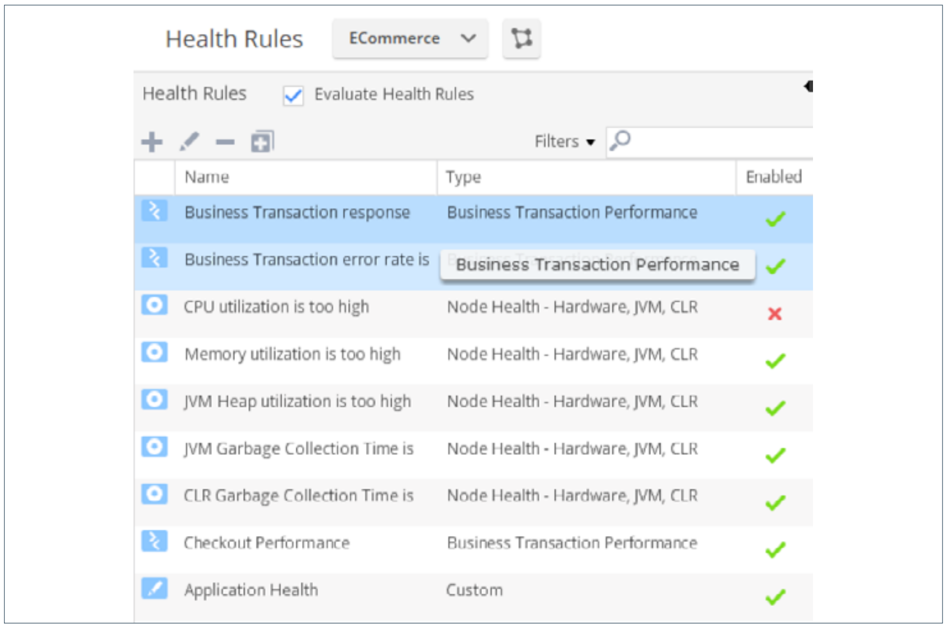
Response Time vs. Load comparisons can again be made using the overall application as a point of reference or individual elements. In this view we are using a 'stacked chart' view of the data series plotted separately over time.


Correlation Analysis
00/14/15 12:00 PM - 00/14/15 4:00 PM
View
Select which Metrics to Correlate
Compare any two metrics by setting the X and Y axis
X Axis
Overall Application Performance | Calls per Minute
Select X Axis Metric
Y Axis
Overall Application Performance | Errors per Minute
Select Y Axis Metric
Errors per Minute vs Calls per Minute
— Best Fit Line (Hide)
Errors per Minute
Calls per Minute
Correlation analysis enables users to select ANY collected metrics for comparison.

Correlation Analysis functions are similar to Scalability analysis above, however, one can select any two measurements for comparison. All measurements collected by AppDynamics are available within this view. Both stacked charts and X-Y graphs are available. In this example the Errors per Min vs Volume are plotted, showing a direct correlation between the increase in Volume and Error Rate.

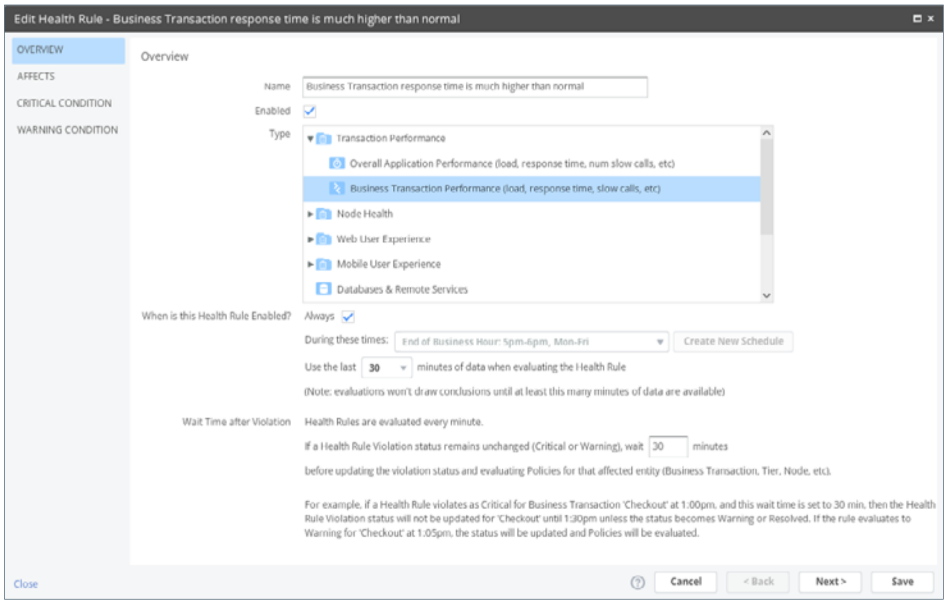
Health rules and alerts

AppDynamics contains powerful capabilities for monitoring and alerting based on various conditions, including complex matching or a combination of conditions via a simple, modern, web-based interface. Health rules are capable of leveraging simple 'static' thresholds along with automatically-calculated baselines for driving actions and alerts. Performance Engineers can establish health rules, which, in turn, fire alerts to notify engineers during load tests. These alerts typically consist of conditions that cross static or dynamic SLA's, which are calculated against dynamic baseline performance. In addition to notifications, these alerts can generate events, and execute custom actions which allow for rapid analysis or integration into existing toolchains.



Name	Type	Enabled
Business Transaction response	Business Transaction Performance	✓
Business Transaction error rate is	Business Transaction Performance	✓
CPU utilization is too high	Node Health - Hardware, JVM, CLR	✗
Memory utilization is too high	Node Health - Hardware, JVM, CLR	✓
JVM Heap utilization is too high	Node Health - Hardware, JVM, CLR	✓
JVM Garbage Collection Time is	Node Health - Hardware, JVM, CLR	✓
CLR Garbage Collection Time is	Node Health - Hardware, JVM, CLR	✓
Checkout Performance	Business Transaction Performance	✓
Application Health	Custom	✓

A view of current health rules in a default AppDynamics deployment. All applications in AppDynamics have a basic set of health rules that are predefined for simplicity and ease of implementation.



Edit Health Rule - Business Transaction response time is much higher than normal

OVERVIEW
AFFECTS
CRITICAL CONDITION
WARNING CONDITION

Overview

Name: Business Transaction response time is much higher than normal

Enabled: ✓

Type:
- Transaction Performance
  - Overall Application Performance (load, response time, num slow calls, etc)
  - Business Transaction Performance (load, response time, slow calls, etc)
- Node Health
- Web User Experience
- Mobile User Experience
- Databases & Remote Services

When is this Health Rule Enabled? Always ✓

During these times: End of Business Hour: 5pm-6pm, Mon-Fri | Create New Schedule

Use the last 30 minutes of data when evaluating the Health Rule

(Note: evaluations won't draw conclusions until at least this many minutes of data are available)

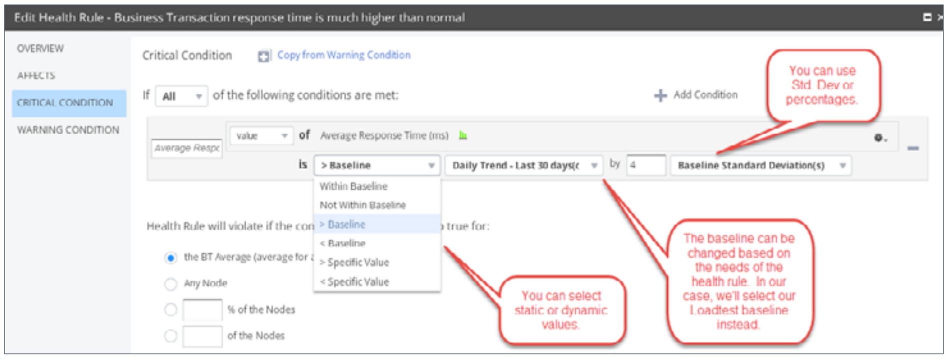
Wait Time after Violation: Health Rules are evaluated every minute.

If a Health Rule Violation status remains unchanged (Critical or Warning), wait 30 minutes

before updating the violation status and evaluating Policies for that affected entity (Business Transaction, Tier, Node, etc).

For example, if a Health Rule violates as Critical for Business Transaction 'Checkout' at 1:00pm, and this wait time is set to 30 min, then the Health Rule Violation status will not be updated for 'Checkout' until 1:30pm unless the status becomes Warning or Resolved. If the rule evaluates to Warning for 'Checkout' at 1:05pm, the status will be updated and Policies will be evaluated.

Defining Health Rules (or modifying existing rules) is a simple, wizard-driven process, allowing users to create new rules based on categories of data classification. A single rule can be created which may be applied to all transactions in the application, for example.



Edit Health Rule - Business Transaction response time is much higher than normal

OVERVIEW
AFFECTS
CRITICAL CONDITION
WARNING CONDITION

Critical Condition | Copy from Warning Condition

If All of the following conditions are met: | Add Condition

Average Respo | value of Average Response Time (ms)

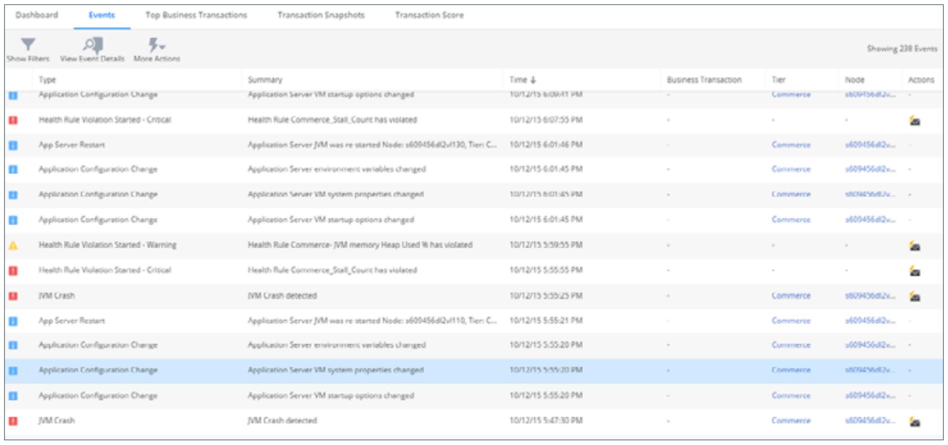
is > Baseline | Daily Trend - Last 30 days(c | by 4 | Baseline Standard Deviation(s)

Health Rule will violate if the con… true for:

- the BT Average (average for …
- Any Node
- % of the Nodes
- of the Nodes

Within Baseline
Not Within Baseline
> Baseline
< Baseline
> Specific Value
< Specific Value
You can use Std. Dev or percentages.
You can select static or dynamic values.
The baseline can be changed based on the needs of the health rule. In our case, we'll select our Loadtest baseline instead.

Setting one or more conditions is simple. Users may choose warning and critical conditions, select multiple measurements, along with applying AND/OR logic. Users may choose static values, or they may choose to utilize the automatically-calculated baselines which takes into account learned behaviors over time.



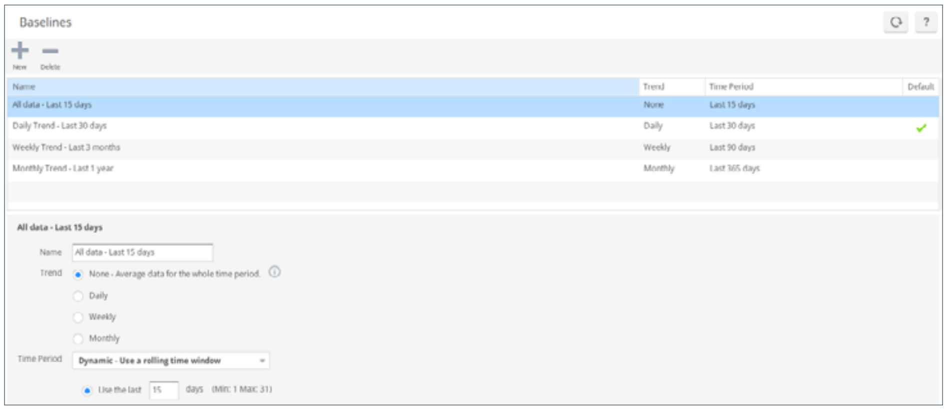
Showing 238 Events

	Type	Summary	Time ↓	Business Transaction	Tier	Node	Actions
🟦	Application Configuration Change	Application Server VM startup options changed	10/12/15 6:09:41 PM	-	Commerce	s609456dl2v...	-
🟥	Health Rule Violation Started - Critical	Health Rule Commerce_Stall_Count has violated	10/12/15 6:07:55 PM	-	-	-	
🟦	App Server Restart	Application Server JVM was re started Node: s609456dl2vl130, Tier: C...	10/12/15 6:01:46 PM	-	Commerce	s609456dl2v...	-
🟦	Application Configuration Change	Application Server environment variables changed	10/12/15 6:01:45 PM	-	Commerce	s609456dl2v...	-
🟦	Application Configuration Change	Application Server VM system properties changed	10/12/15 6:01:45 PM	-	Commerce	s609456dl2v...	-
🟦	Application Configuration Change	Application Server VM startup options changed	10/12/15 6:01:45 PM	-	Commerce	s609456dl2v...	-
⚠	Health Rule Violation Started - Warning	Health Rule Commerce- JVM memory Heap Used % has violated	10/12/15 5:59:55 PM	-	-	-	
🟥	Health Rule Violation Started - Critical	Health Rule Commerce_Stall_Count has violated	10/12/15 5:55:55 PM	-	-	-	
🟥	JVM Crash	JVM Crash detected	10/12/15 5:55:25 PM	-	Commerce	s609456dl2v...	
🟦	App Server Restart	Application Server JVM was re started Node: s609456dl2vl110, Tier: C...	10/12/15 5:55:21 PM	-	Commerce	s609456dl2v...	-
🟦	Application Configuration Change	Application Server environment variables changed	10/12/15 5:55:20 PM	-	Commerce	s609456dl2v...	-
🟦	Application Configuration Change	Application Server VM system properties changed	10/12/15 5:55:20 PM	-	Commerce	s609456dl2v...	-
🟦	Application Configuration Change	Application Server VM startup options changed	10/12/15 5:55:20 PM	-	Commerce	s609456dl2v...	-
🟥	JVM Crash	JVM Crash detected	10/12/15 5:47:30 PM	-	Commerce	s609456dl2v...	

Once Health Rules have been configured, they will produce events which may be easily reviewed for additional detail, along with notification to the appropriate user or users.

User-configurable baselines

Auto-baselining is a standard core platform feature within AppDynamics. It allows AppDynamics to dynamically learn what's 'normal' for an application over time. Baselines are not just calculated for the overall application, but for each and every metric collected. This means each transaction, each CPU measurement, along with every single metric collected via the core platform or extensions are baselined. This frees up time among Operations users, but is less applicable to performance engineering needs. To facilitate other use-cases, users may configure baseline behaviors to meet the needs of performance engineering use cases as well. Configuring baselines specific to load tests enables performance engineers to maximize the usefulness of a baseline by eliminating 'white space' from the timeline, e.g. gaps in traffic in the performance environment between load tests. The result is a baseline calculation that focuses only on data collected during an active load test. As a result, specific load tests may be used as a standard against which other load tests are compared.



Baselines

Name	Trend	Time Period	Default
All data - Last 15 days	None	Last 15 days	
Daily Trend - Last 30 days	Daily	Last 30 days	✓
Weekly Trend - Last 3 months	Weekly	Last 90 days	
Monthly Trend - Last 1 year	Monthly	Last 365 days	

All data - Last 15 days

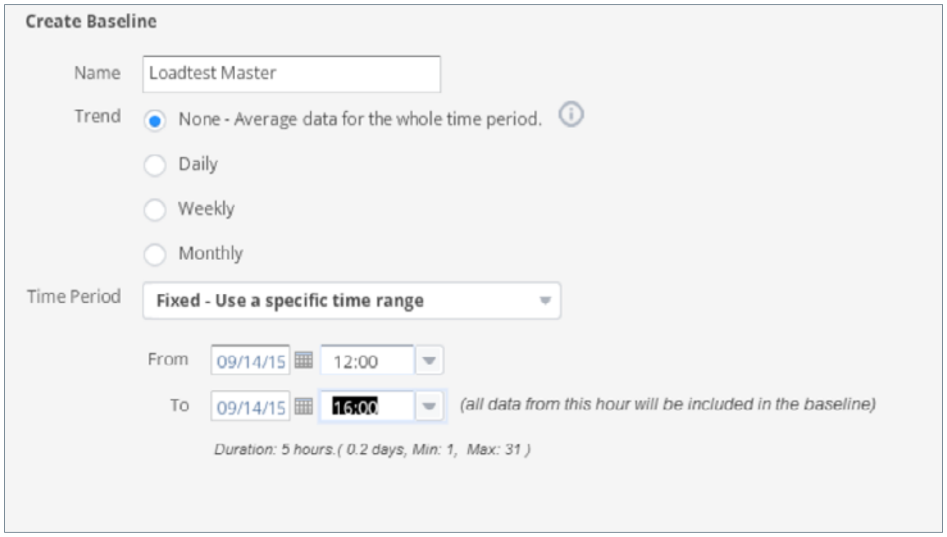
Name: All data - Last 15 days

Trend: [✓ None - Average data for the whole time period.] Daily Weekly Monthly

Time Period: Dynamic - Use a rolling time window

[✓ Use the last] 15 days (Min: 1 Max: 31)

Defining and selecting a baseline is simple. Multiple baselines may be selected, as they are calculated separately from the data that AppDynamics collects.



Create Baseline

Name: Loadtest Master

Trend: [✓ None - Average data for the whole time period.] Daily Weekly Monthly

Time Period: Fixed - Use a specific time range

From 09/14/15 12:00

To 09/14/15 16:00 *(all data from this hour will be included in the baseline)*

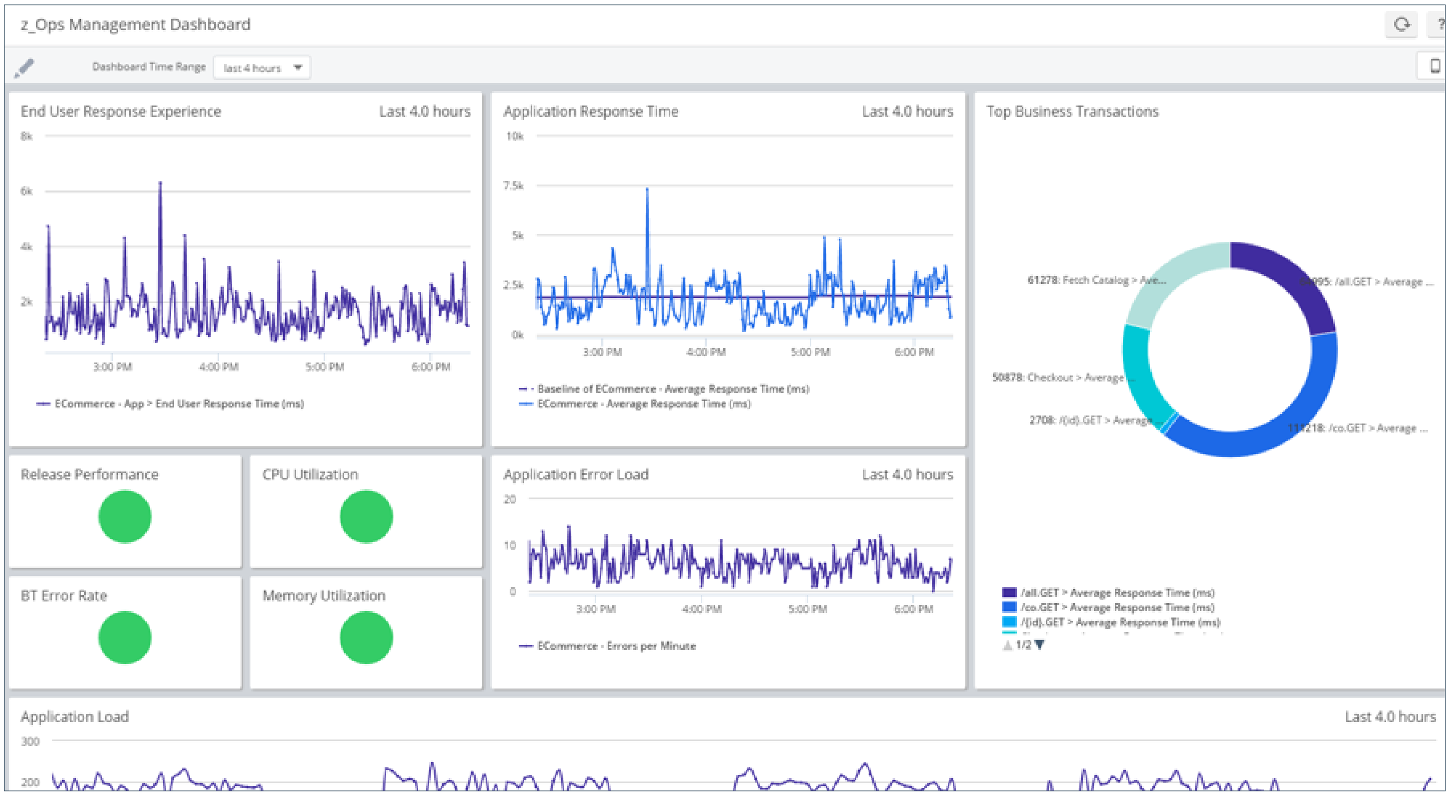
Duration: 5 hours.(0.2 days, Min: 1, Max: 31)

Similar to custom time ranges, baselines simply need to map to a load test's timeframe.

Custom dashboards

AppDynamics provides powerful, flexible dashboarding which can be used to visualize any data collected by AppDynamics. These dashboards display data in real time, but can also display historical data along with producing PDF reports. In addition, these dashboards are exportable, reusable, and leverage custom baselines, health rules, and time ranges. Performance Engineers use dashboarding for these typical scenarios:

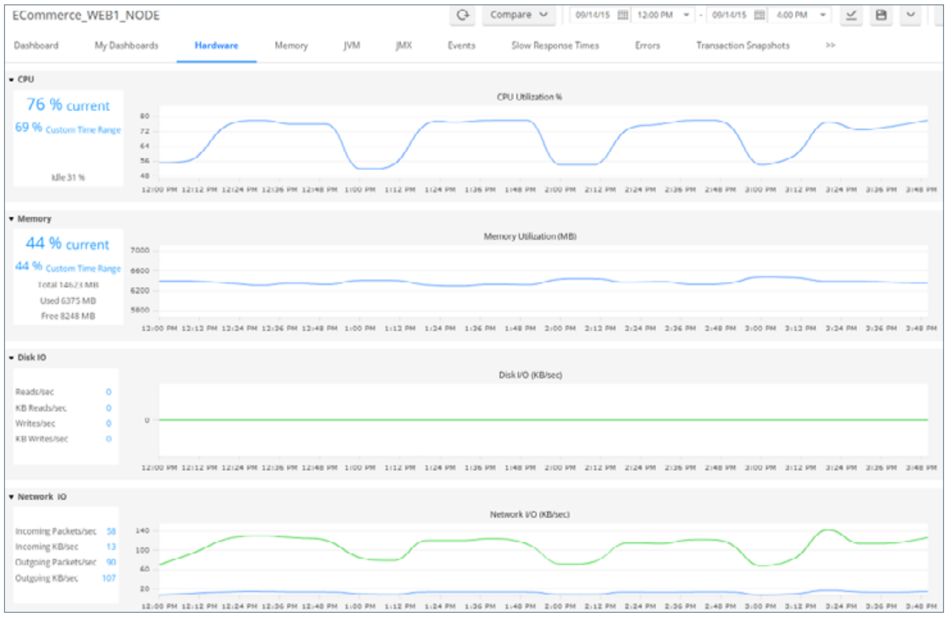
- Before running a performance test, dashboards for a given application environment provide an at-a-glance view to determine if the environment is running, healthy, and ready for testing.
- Dashboards provide detailed visibility and insight into the application during load test.
- Component-level dashboards may be built and re-used for any number of technologies, for example Apache, WebLogic, WebSphere, IIS, Databases, and more.



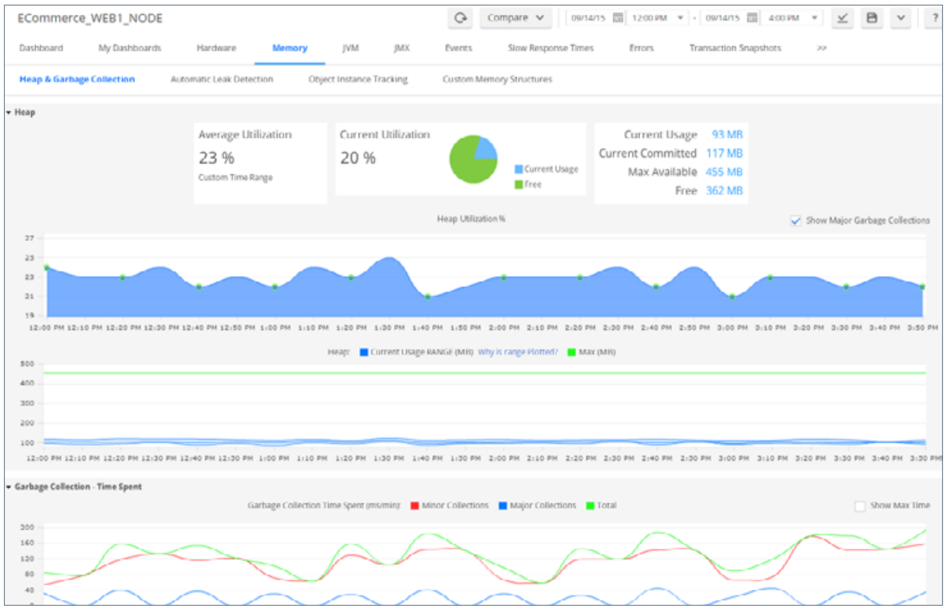
z_Ops Management Dashboard
Dashboard Time Range
last 4 hours
End User Response Experience
Last 4.0 hours
8k
6k
4k
2k
3:00 PM
4:00 PM
5:00 PM
6:00 PM
ECommerce - App > End User Response Time (ms)
Application Response Time
Last 4.0 hours
10k
7.5k
5k
2.5k
0k
Baseline of ECommerce - Average Response Time (ms)
ECommerce - Average Response Time (ms)
Top Business Transactions
61278: Fetch Catalog > Ave...
64995: /all.GET > Average ...
50878: Checkout > Average ...
2708: /{id}.GET > Average ...
111218: /co.GET > Average ...
/all.GET > Average Response Time (ms)
/co.GET > Average Response Time (ms)
/{id}.GET > Average Response Time (ms)
1/2
Release Performance
CPU Utilization
BT Error Rate
Memory Utilization
Application Error Load
Last 4.0 hours
20
10
0
ECommerce - Errors per Minute
Application Load
Last 4.0 hours
300
200

Detailed infrastructure metrics

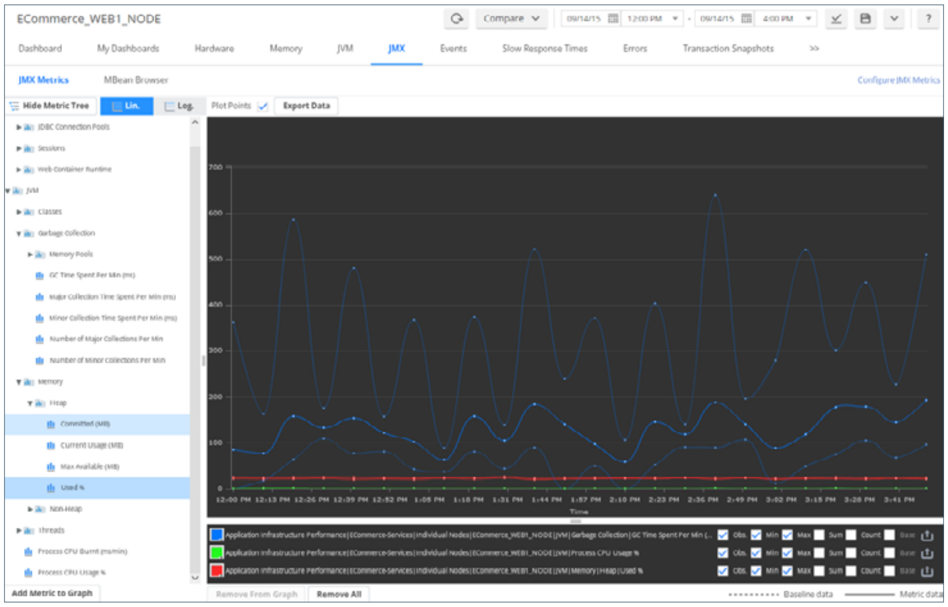
AppDynamics automatically collects a large amount of application and infrastructure data including Operating System metrics, JVM metrics, .NET metrics, PHP metrics, Apache metrics, Node.js metrics, Python metrics, and JMX metrics, simply by enabling agents. This data is then immediately usable within dashboards, health rules which in turn can be used for alerting or custom actions, comparisons and other analytics without need for additional configuration. This eliminates the extra configuration steps in HP LoadRunner, HP Performance Center and/or HP SiteScope which are necessary when trying to collect additional metrics.



Out of the box Operating System metrics.



Detailed JVM and heap information, including heap used, garbage collections information and timing along with detailed heap breakdown.



🟦 Application Infrastructure Performance|ECommerce-Services|Individual Nodes|ECommerce_WEB1_NODE|JVM|Garbage Collection|GC Time Spent Per Min (... ✓ Obs. ✓ Min ✓ Max Sum Count Base

🟩 Application Infrastructure Performance|ECommerce-Services|Individual Nodes|ECommerce_WEB1_NODE|JVM|Process CPU Usage % ✓ Obs. ✓ Min ✓ Max Sum Count Base

🟥 Application Infrastructure Performance|ECommerce-Services|Individual Nodes|ECommerce_WEB1_NODE|JVM|Memory|Heap|Used % ✓ Obs. ✓ Min ✓ Max Sum Count Base

The metric browser enables users to access metric data collected and quickly/chart graph the data for ad-hoc analysis.

Conclusion and summary

AppDynamics has a number of features and functions which not only help production deployments, but also add new visibility and analytics for those doing performance engineering. This document focuses on LoadRunner-specific scenarios, but most of the topics may be applied to most any load testing solution on the market. The critical key benefits are as follows:

- AppDynamics can easily be configured to show LoadRunner transactions in the way that LoadRunner users see them today.
- Flexible dashboards are not only reusable but enable historical and real-time analysis. This benefits users with a high degree of visibility while reducing configuration effort, saving time and manpower.
- Automatic baselining allows rapid comparative-analysis against a specific ‘master’ performance profile. This results in more realistic loadtesting analysis and higher-quality results by measuring against actuals vs. ‘guesstimates’ for expected performance.
- Built-in comparison functionality between two different load tests facilitates rapid analysis and reduced effort to answer questions including ‘what changed?’ or ‘did performance improve between releases?’
- Deep Code-level visibility provides performance engineers the ability to quick triage bottlenecks in the application, minimizing the need for developers during the analysis phase. This keeps developers focused on building new functionality while reducing the time that performance engineers spend pinpointing problems.
- Deep Database visibility offers contextualized database performance breakdowns to further reduce root-cause analysis times and provide details in the same ‘language’ that DBA’s use.
- Real-time alerting during load tests enables notification of breached SLAs or deviations from expected results. This saves performance engineers time and allows for more rapid analysis of issues by surfacing errors more quickly.
- Automatically-configured, out-of-the-box metrics reduce configuration time and deployment effort, as well as ongoing maintenance.

Try it FREE at appdynamics.com