Collecting Baselines for APM – Finding Out Which Metrics Matter

Michael Sydor

SERVICE ASSURANCE - APM

Table of Contents

Copyright ©2010 CA. All rights reserved. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies. This document is for your informational purposes only. CA assumes no responsibility for the accuracy or completeness of the information. To the extent permitted by applicable law, CA provides this document "as is" without warranty of any kind, including, without limitation, any implied warranties of merchantability, fitness for a particular purpose, or noninfringement. In no event will CA be liable for any loss or damage, direct or indirect, from the use of this document, including, without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised in advance of the possibility of such damages.

Executive Summary

Challenge

Successful use of APM technology requires effective skills and processes. This series of tech notes explores a number of topics that you should consider, when using APM.

Whenever you encounter a performance problem, the first question you ask yourself is "what changed"? To appreciate what has changed requires that you know *what is normal*, for your application. You define normal by collecting characteristics into a simple report called a *baseline*.

Baselines are the foundation for constructing effective dashboards based on thresholds and alerts, as well as tracking the quality of your software over the application life cycle.

Opportunity

Collecting baselines exercises core techniques for using the APM workstation to quickly identify the significant components of your application, defining metric groupings and assembling a report and dashboard. Generating these baselines, as part of the application lifecycle, means that you will always have a definition of normal, for your application. This will also help you to design and construct effective dashboards so that casual users will have the benefit of what you have learned about the application, and without having to repeat your work.

Benefits

Collecting baselines gives you the following benefits:

- Effective use of the APM workstation
- Understanding the key components of your application
- More efficient triage of performance problems
- Better collaboration with your stakeholders.

SECTION 1: CHALLENGE

Which Metrics Matter?

Applications are always undergoing some kind of change. It may be a new code release. It may be a refresh of the network or hardware. When something changes your application performance may be at risk.

Baselines are the process to help you manage that change by helping you detect what components in your application are affected. Until you know exactly what is being affected you do not know where to make the adjustments that will bring your application back to the performance level you want.

Finding a normal period

To get a good baseline you need to have a representative period where the activity of the application meets your criteria for “normal”. This is almost never when you are having a performance problem. It has to be a period before the problem occurred. Exactly how much earlier is up to you but the earlier, the better.

You will apply the same report template to the incident, as you do to the baseline. But the baseline has to come first.

If your application is already deployed in production, then a production baseline is what you will collect. This has some challenges, as you will so find out. The best time to collect baselines is when you are stress testing the application in the QA environment. This also has some challenges but this environment gives us the best chance of defining “normal” for you application.

QA BASELINE

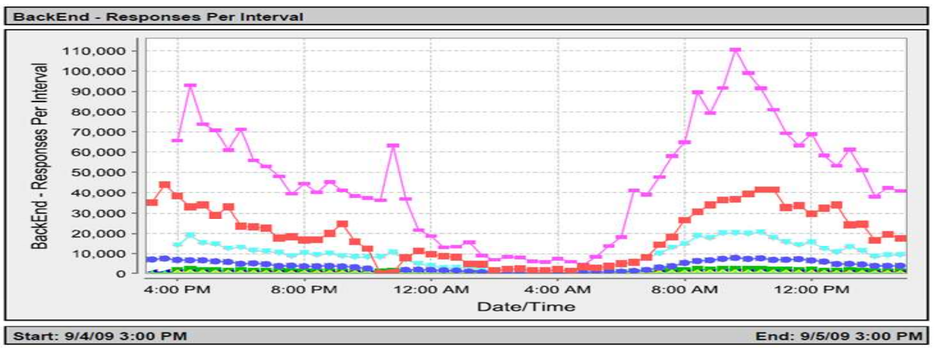
To collect a baseline in QA you simply need to have APM enabled on the application before you start running your testing. A simulated load is best and you want to have a consistent period of load, called steady-state, where you will focus your baseline efforts.

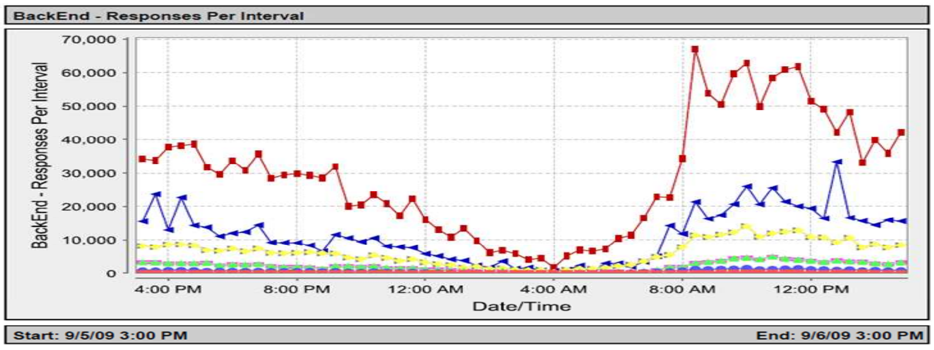
You can also collect baselines with manual load but that load activity needs to be consistent. The more consistent the load profile (manual or automated), to more accurate the baseline will be.

To confirm that delivery of your load is consistent, or *reproducible*, you want to execute a single load profile for three consecutive runs of testing. When you analyze these results, you should get the same values for response times, number of invocations, etc.

PRODUCTION BASELINE

A production environment may also be used for collecting the baselines but it can be very challenging to find a consistent, representative period. The following figure illustrates this challenge.:

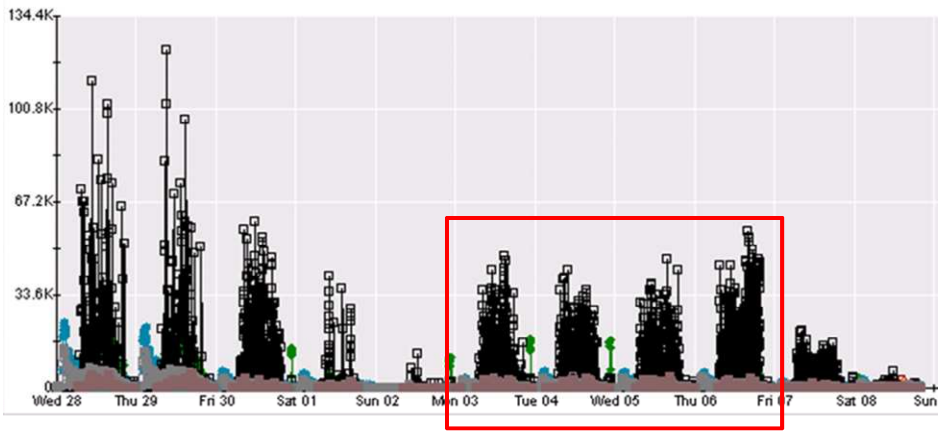




As you can see in the two charts, the curves are roughly equivalent. These are some deviations but over the 24 hour period these two historical views have the same characteristics. That is until you look at the actual responses per interval. When you look at the uppermost curve, in each chart, the top chart varies from

90,000 to 110,000 responses per interval , while the lower chart varies from 40,000 to 70,000 responses per interval. Which one in the normal level of activity? You cannot really decide until you look at more sample periods.

Here is another example, this time showing an acceptable level of variation. This is also for a production baseline where it was decided that the indicated 4 days would define normal. The prior week had a higher level of interaction but the application, in this case, was changed over the weekend. So the new activity levels now constituted "normal" for this application.



When you establish a baseline, it is not a static definition. The baseline needs to be validated with every change in the application or its operating environment. "Normal", for your application, is probably a moving target. It is an exercise that can be accomplished in a few minutes. If nothing has changed in the baseline, with respect to the components that best represent performance, then the baseline definition doesn't need to change. If new key components are encountered or different levels of performance expereinced, then these changes need to be accommodated in the baseline template definition and in the alert thresholds.

When a baseline changes, it is a call to action because metric groups, thresholds, reports and dashboards and alerts, are all dependent on the baseline information. If you want to keep all of that information correct, the baseline is how you will know that you monitoring configuration is correct and valid.

Key Metrics

When you measure the performance of application components you get a number of metrics for each component. The basic five metrics are response time, responses per interval (invocations), concurrency (concurrent invocations), errors and stalls. Two of these are essential to start an effective baseline: response time and invocations. The other metrics are also valuable but you should consider them once your baseline practice is established. For now, just looking at these two metrics will get you off to a solid start with baselines.

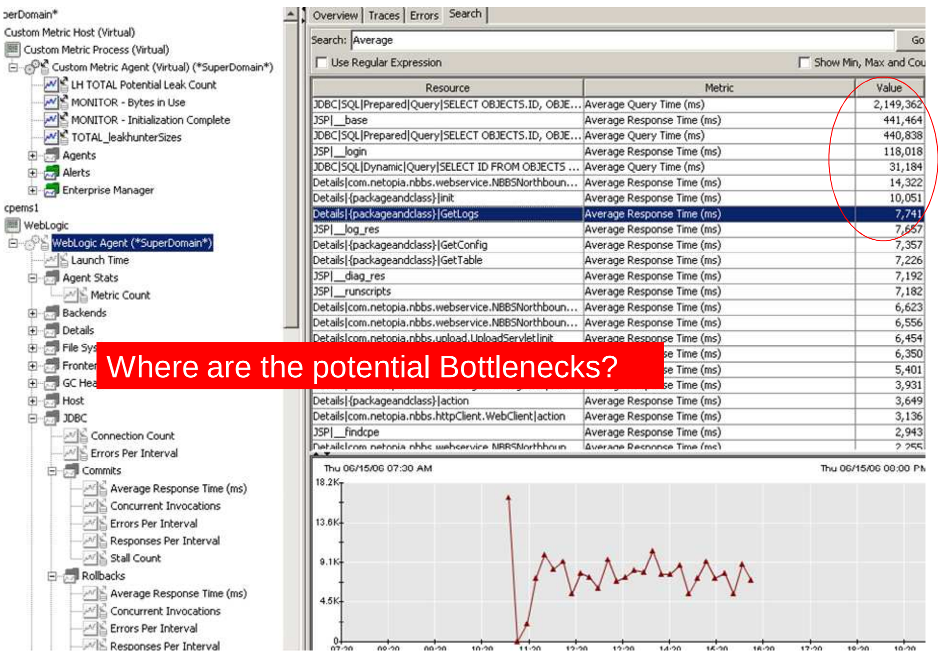
If you are new to APM you may have noticed that thousands of metrics are being collected for your, 24 hours by 7 days, these metrics are being stored for you to use in addressing performance issues. It seems like a mountain of information but the techniques we will discuss today will get you quickly focused on the top 10 or 20 metrics that best represent your application performance.

Eventually, you will use these baselines to define threshold and alerts, and set the foundation for SLA's and other service management activities, as well as managing performance along you application life cycle.

RESPONSE TIME EXAMPLE

The response time metrics shows you which components are the potential bottlenecks for you application. The activities that you spend lots of computer time on are real opportunities to make the application more efficient. Which components and queries does your application spend the most time on?

The following figure shows the explorer view of the APM workstation, with the Search Tab selected.



Search: Average

☐ Use Regular Expression ☐ Show Min, Max and Cou

Resource	Metric	Value
JDBC|SQL|Prepared|Query|SELECT OBJECTS.ID, OBJE...	Average Query Time (ms)	2,149,362
JSP|__base	Average Response Time (ms)	441,464
JDBC|SQL|Prepared|Query|SELECT OBJECTS.ID, OBJE...	Average Query Time (ms)	440,838
JSP|__login	Average Response Time (ms)	118,018
JDBC|SQL|Dynamic|Query|SELECT ID FROM OBJECTS ...	Average Query Time (ms)	31,184
Details|com.netopia.nbbs.webservice.NBBSNorthboun...	Average Response Time (ms)	14,322
Details|{packageandclass}|init	Average Response Time (ms)	10,051
Details|{packageandclass}|GetLogs	Average Response Time (ms)	7,741
JSP|__log_res	Average Response Time (ms)	7,657
Details|{packageandclass}|GetConfig	Average Response Time (ms)	7,357
Details|{packageandclass}|GetTable	Average Response Time (ms)	7,226
JSP|__diag_res	Average Response Time (ms)	7,192
JSP|__runscripts	Average Response Time (ms)	7,182
Details|com.netopia.nbbs.webservice.NBBSNorthboun...	Average Response Time (ms)	6,623
Details|com.netopia.nbbs.webservice.NBBSNorthboun...	Average Response Time (ms)	6,556
Details|com.netopia.nbbs.upload.UploadServlet|init	Average Response Time (ms)	6,454
[illegible]	se Time (ms)	6,350
[illegible]	se Time (ms)	5,401
[illegible]	se Time (ms)	3,931
Details|{packageandclass}|action	Average Response Time (ms)	3,649
Details|com.netopia.nbbs.httpClient.WebClient|action	Average Response Time (ms)	3,136
JSP|__findcpe	Average Response Time (ms)	2,943
Details|com.netopia.nbbs.webservice.NBBSNorthboun	Average Response Time (ms)	2,255

On the left hand side an agent has been selected. Underneath that agent are all of the components reporting. Some of these components are expanded to review the individual metrics underneath each unique component. Navigation among these metrics is tedious when you don't know what you are looking for. Using the search tab eliminates the need for any navigation.

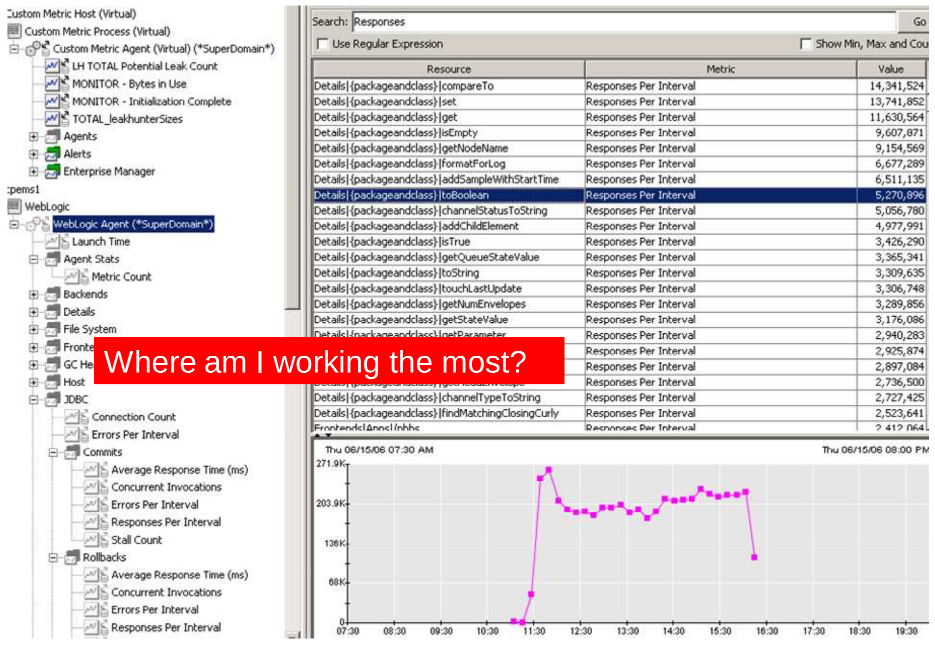
There is a search field into which you type the word "Average" (no quotes). When you hit return, the APM workstation will then show all of the metrics containing the word "Average", for the current data or historical range. You then click on the "Value" header to cause the results to be sorted, from greatest to least. If you click on any of the metric rows, you will get a graph below, for the range you are looking at. You can also <shift>><click> to add multiple metrics to the same graph.

You might also notice that the query match both response times and query times. You can keep these metric names separate but it doesn't matter at this point. Whatever shows up here is simply the biggest of response times.

The next step is to define a metric group and add the top 10 or twenty metrics. The name of this metric group should be PERFORMANCE – that's just what it means. That name will make it easy to find later when you are defining alerts, reports and dashboards.

INVOCATIONS EXAMPLE

The invocations metrics (Response per Interval) lets you know where your application is the most active. This provides a precise mechanism to measure and track capacity. When the application is saturated, where it can no longer scale, the invocations tell you exactly how many key interactions are being performed.



Search: Responses Go

☐ Use Regular Expression ☐ Show Min, Max and Cou

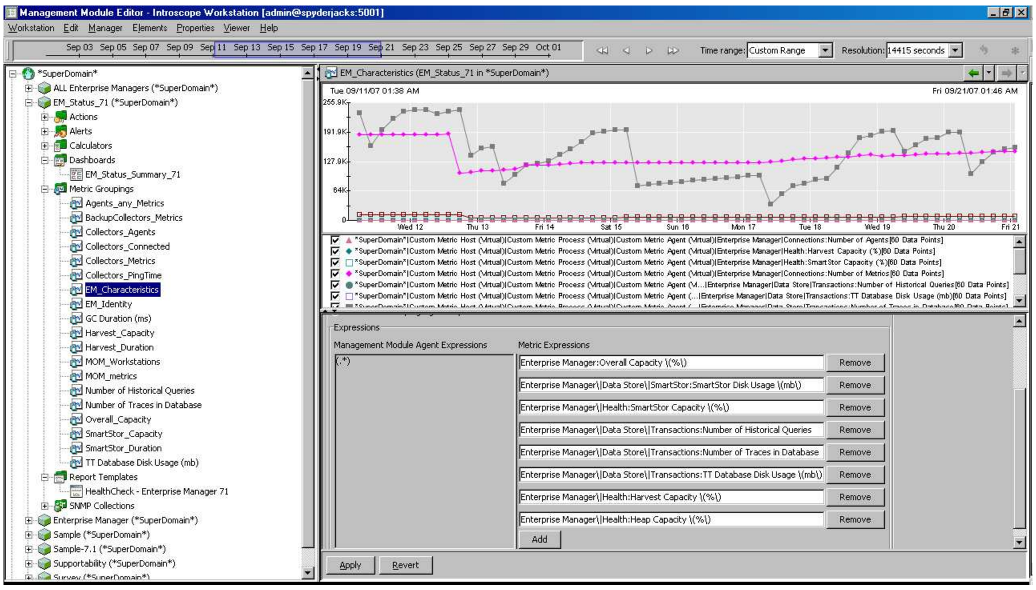
Resource	Metric	Value
Details|{packageandclass}|compareTo	Responses Per Interval	14,341,524
Details|{packageandclass}|set	Responses Per Interval	13,741,852
Details|{packageandclass}|get	Responses Per Interval	11,630,564
Details|{packageandclass}|isEmpty	Responses Per Interval	9,607,871
Details|{packageandclass}|getNodeName	Responses Per Interval	9,154,569
Details|{packageandclass}|formatForLog	Responses Per Interval	6,677,289
Details|{packageandclass}|addSampleWithStartTime	Responses Per Interval	6,511,135
Details|{packageandclass}|toBoolean	Responses Per Interval	5,270,896
Details|{packageandclass}|channelStatusToString	Responses Per Interval	5,056,780
Details|{packageandclass}|addChildElement	Responses Per Interval	4,977,991
Details|{packageandclass}|isTrue	Responses Per Interval	3,426,290
Details|{packageandclass}|getQueueStateValue	Responses Per Interval	3,365,341
Details|{packageandclass}|toString	Responses Per Interval	3,309,635
Details|{packageandclass}|touchLastUpdate	Responses Per Interval	3,306,748
Details|{packageandclass}|getNumEnvelopes	Responses Per Interval	3,289,856
Details|{packageandclass}|getStateValue	Responses Per Interval	3,176,086
Details|{packageandclass}|getParameter	Responses Per Interval	2,940,283
[illegible]	Responses Per Interval	2,925,874
[illegible]	Responses Per Interval	2,897,084
[illegible]	Responses Per Interval	2,736,500
Details|{packageandclass}|channelTypeToString	Responses Per Interval	2,727,425
Details|{packageandclass}|findMatchingClosingCurly	Responses Per Interval	2,523,641
Frontends|Apps|/pbbs	Responses Per Interval	2,412,064

The sequence of operations is exactly the same as before, with the exception of the Search field, which is now equal to "Responses". After sorting the Value column, verify your top 10 or 20 metrics, create a new metric group called CAPACITY, and drag the metrics into that new metric group.

Building Metric Groups

To be efficient, you want to drag-n-drop metrics, instead of typing them. This requires that you have to APM workstation windows open: an Explorer and Management Module Editor. You create the management module, or navigate to it if it is already existing and then drag metrics from the Explorer window and drop into the fields of the metric grouping.

The following figure is an example, this time with Enterprise Manager metrics. You don't have to wait for a complex application to try out your workstation skills. You always have the Enterprise Manager metrics around for practice.


Management Module Editor - Introscope Workstation [admin@spyderjacks:5001]
Workstation Edit Manager Elements Properties Viewer Help
Sep 03 Sep 05 Sep 07 Sep 09 Sep 11 Sep 13 Sep 15 Sep 17 Sep 19 Sep 21 Sep 23 Sep 25 Sep 27 Sep 29 Oct 01
Time range:
Resolution:
EM_Characteristics (EM_Status_71 in *SuperDomain*)
Tue 09/11/07 01:38 AM
Fri 09/21/07 01:46 AM
255.9K
191.9K
127.9K
64K
0
Wed 12 Thu 13 Fri 14 Sat 15 Sun 16 Mon 17 Tue 18 Wed 19 Thu 20 Fri 21
Expressions
Management Module Agent Expressions
Metric Expressions
Remove
Add
Apply
Revert

Time range: Custom Range

Resolution: 14415 seconds

```
*SuperDomain*
  ALL Enterprise Managers (*SuperDomain*)
  EM_Status_71 (*SuperDomain*)
    Actions
    Alerts
    Calculators
    Dashboards
      EM_Status_Summary_71
    Metric Groupings
      Agents_any_Metrics
      BackupCollectors_Metrics
      Collectors_Agents
      Collectors_Connected
      Collectors_Metrics
      Collectors_PingTime
      EM_Characteristics
      EM_Identity
      GC Duration (ms)
      Harvest_Capacity
      Harvest_Duration
      MOM_Workstations
      MOM_metrics
      Number of Historical Queries
      Number of Traces in Database
      Overall_Capacity
      SmartStor_Capacity
      SmartStor_Duration
      TT Database Disk Usage (mb)
    Report Templates
      HealthCheck - Enterprise Manager 71
    SNMP Collections
  Enterprise Manager (*SuperDomain*)
  Sample (*SuperDomain*)
  Sample-7.1 (*SuperDomain*)
  Supportability (*SuperDomain*)
  Survey (*SuperDomain*)
```

```
☑ ▲ *SuperDomain*|Custom Metric Host (Virtual)|Custom Metric Process (Virtual)|Custom Metric Agent (Virtual)|Enterprise Manager|Connections:Number of Agents[60 Data Points]
☑ ◆ *SuperDomain*|Custom Metric Host (Virtual)|Custom Metric Process (Virtual)|Custom Metric Agent (Virtual)|Enterprise Manager|Health:Harvest Capacity (%)[60 Data Points]
☑ □ *SuperDomain*|Custom Metric Host (Virtual)|Custom Metric Process (Virtual)|Custom Metric Agent (Virtual)|Enterprise Manager|Health:SmartStor Capacity (%)[60 Data Points]
☑ ◆ *SuperDomain*|Custom Metric Host (Virtual)|Custom Metric Process (Virtual)|Custom Metric Agent (Virtual)|Enterprise Manager|Connections:Number of Metrics[60 Data Points]
☑ ● *SuperDomain*|Custom Metric Host (Virtual)|Custom Metric Process (Virtual)|Custom Metric Agent (V...|Enterprise Manager|Data Store|Transactions:Number of Historical Queries[60 Data Points]
☑ □ *SuperDomain*|Custom Metric Host (Virtual)|Custom Metric Process (Virtual)|Custom Metric Agent (...|Enterprise Manager|Data Store|Transactions:TT Database Disk Usage (mb)[60 Data Points]
```

Management Module Agent Expressions: (.*)

Metric Expressions:

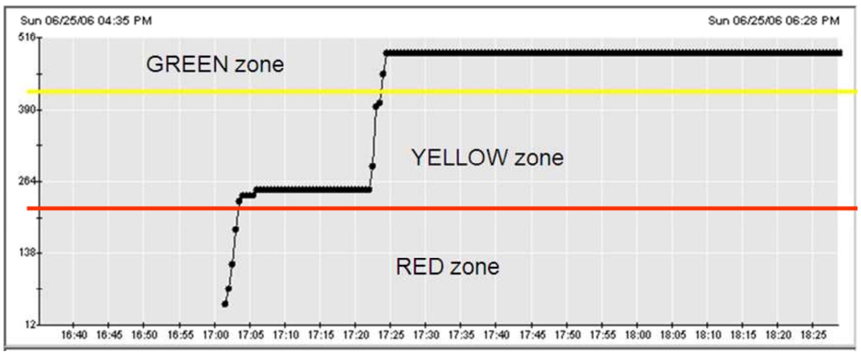
```
Enterprise Manager:Overall Capacity \(%\)
Enterprise Manager\|Data Store\|SmartStor:SmartStor Disk Usage \(mb\)
Enterprise Manager\|Health:SmartStor Capacity \(%\)
Enterprise Manager\|Data Store\|Transactions:Number of Historical Queries
Enterprise Manager\|Data Store\|Transactions:Number of Traces in Database
Enterprise Manager\|Data Store\|Transactions:TT Database Disk Usage \(mb\)
Enterprise Manager\|Health:Harvest Capacity \(%\)
Enterprise Manager\|Health:Heap Capacity \(%\)
```

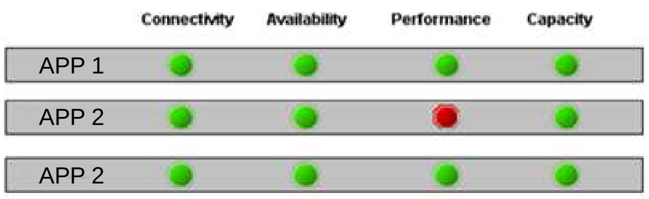
The left pane lists the current metric groupings. The right pane, with the graph and parameter entry fields, holds the current members of the collection. While drag-n-drop is the easiest to start with you may also use a regular expression to define the metrics you are interested in, for the metric grouping.

Once a metric group is established, you are now ready to define thresholds, and build dashboards and reports.

Applying Thresholds

To define a threshold, simply select a metric group and right-click to create a new alert via the pop-up menu. There are a lot of options available for configuration. What you need to focus is the threshold definition which layered over the current values of the elements of your metric group. You cannot set thresholds reliably until your metric group is defined and presenting data. An example result is in the following figure:





The graph is of the metric count and thresholds have been defined so that the various regions will be reflected in an alert lamp. A sample dashboard is directly below the graph for added emphasis of this relationship. The alert definition does not build this dashboard – you need to complete it.

For the metric count thresholds, when the application server starts, it will progress up to a certain point and then it is ready for load. Prior to that point it is "unavailable". It is a feature of Java that the application components are not loaded until users start exercising the application. So if the app server is restarted at 4am but no users arrive until 8am, the app server is actually is an "unproven" state. Since some application servers can hang at startup, this is a useful state to reveal. And the "yellow" means exactly that – "caution – this server has not fully initialized"! When even a single user exercises the application, the remaining Java components get invoked, and the server is now fully initialized – we are in the green state.

Using the metric count is practically airtight, in terms of getting some experience with thresholding and learning a small tidbit about the nuances of your application startup. Everything you do to get this alert established is exactly the same process for the PERFORMANCE and CAPACITY metric groupings - only the level of effort changes.

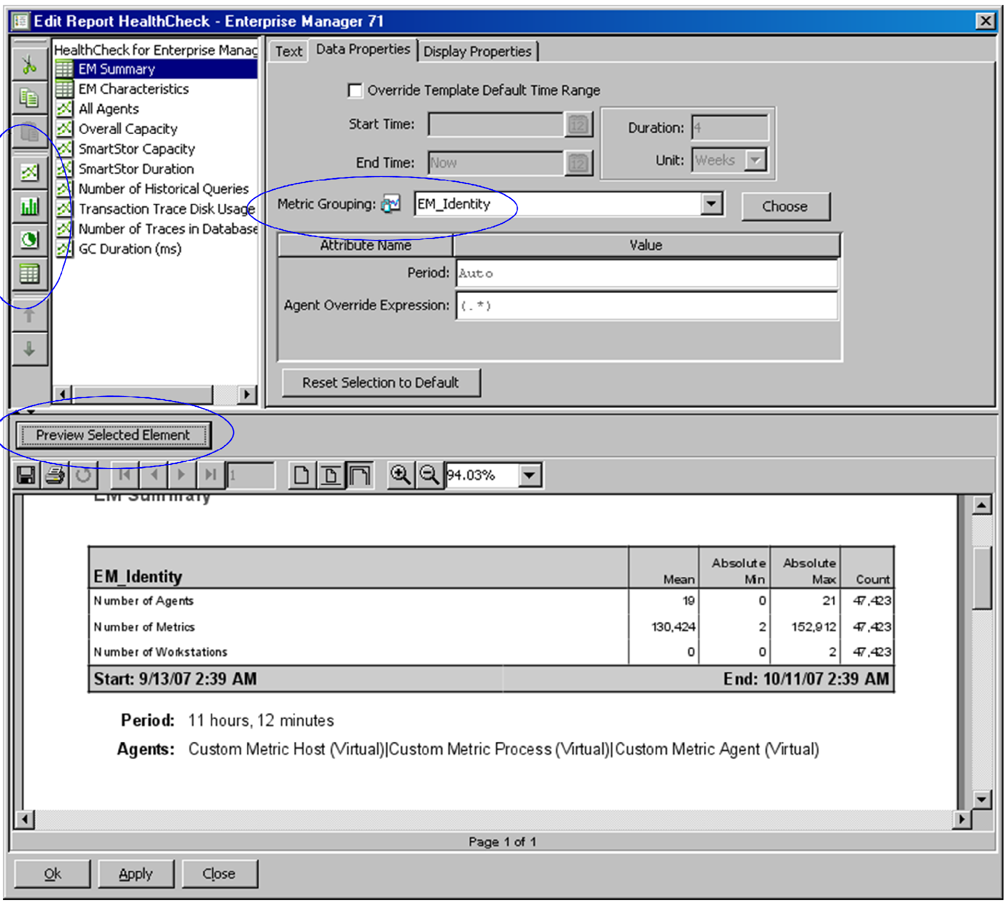
This little dashboard snippet should be interesting for you are well. This is the first dashboard you should attempt because it is heavy on the metric groupings and easy to validate. Sure, everybody wants to build a GIS[1] application for their first dashboard but you really need to "keep it simple". There is nothing worse than a dashboard that looks pretty – but doesn't represent reality.

---

[1] Global Information System http://en.wikipedia.org/wiki/GIS

Building the Report

The other immediate application of a metric group is to define a report template. The following figure shows the definition of a baseline report for the Enterprise Manager. As before, if you do not already have an application to practice on, the EM is always there for you. And like the definition of the metric grouping, you can incrementally test the template to make sure you are getting what you want. The metric group and type of report element are also highlighted.



Override Template Default Time Range

Start Time:

End Time: Now

Duration: 4

Unit: Weeks

Metric Grouping: EM_Identity

Attribute Name	Value
Period:	Auto
Agent Override Expression:	(.*)

EM Summary

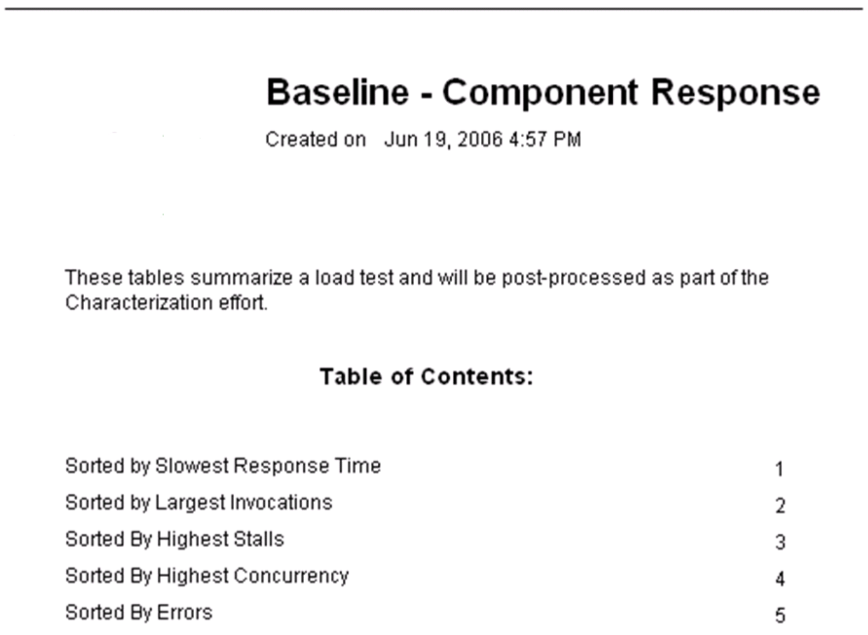
EM_Identity	Mean	Absolute Min	Absolute Max	Count
Number of Agents	19	0	21	47,423
Number of Metrics	130,424	2	152,912	47,423
Number of Workstations	0	0	2	47,423
Start: 9/13/07 2:39 AM				End: 10/11/07 2:39 AM

Period: 11 hours, 12 minutes

Agents: Custom Metric Host (Virtual)|Custom Metric Process (Virtual)|Custom Metric Agent (Virtual)

It will take a couple of minutes to finish the report template definition. And then a couple of test runs to make sure it is ready. But once it is – it is ready for everybody and you have established the first element of collaboration – a consistent presentation of performance information to share with you stakeholders.

To turn back to the performance baseline for your application, after you have mastered the first two key metrics, you can move on and define the remainder. In the next figure is the cover sheet for the full baseline report:



Baseline - Component Response

Created on Jun 19, 2006 4:57 PM

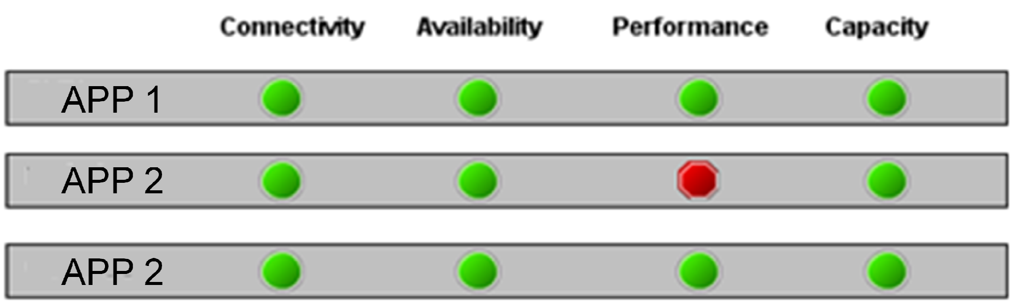
These tables summarize a load test and will be post-processed as part of the Characterization effort.

Table of Contents:

The immediate benefit of doing baselines via report is that you don't need any time at the APM workstation, going through all of those manual operations. You could even reduce the report generation to a simple script.

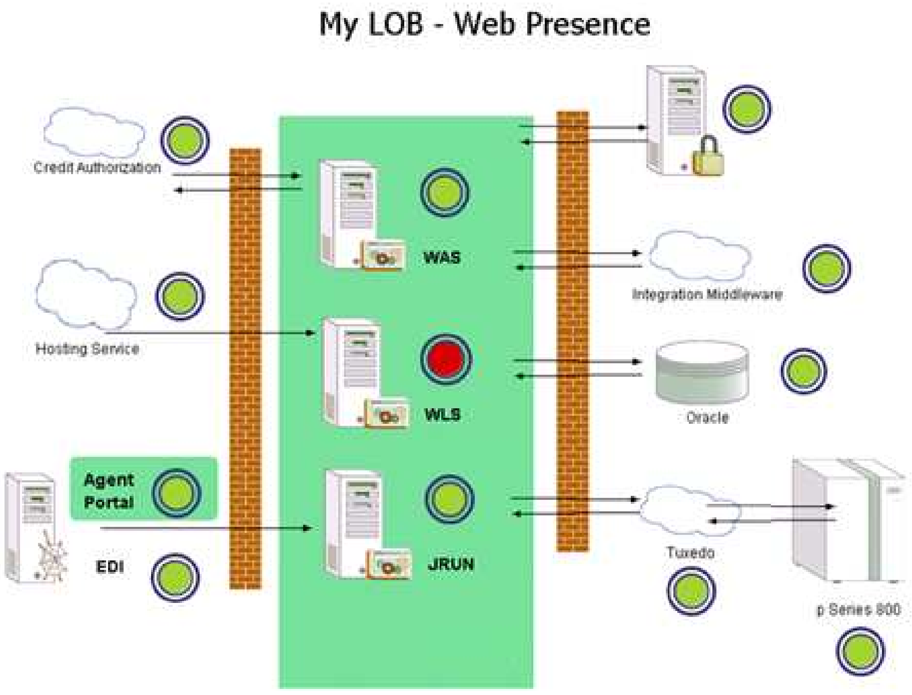
Building the Dashboard

After the baseline report is established, it is time to turn more attention to the dashboard. The dashboards that is easiest to understand, by a wide variety of stakeholders, is the alert-centric dashboard indicated below:



The first two thresholds, CONNECTIVITY and AVAILABILITY are the easiest to define. There is a single agent connection metric and metric count, per agent. So these are a metric group of 1 member, unless the application has multiple instances and you could collect all of the instances together in the metric grouping. The PERFORMANCE and CAPACITY thresholds can have any number of members in their representative metric groupings. These will take more time to confirm that you have the key metrics and to validate that the thresholds are correct. But this dashboard, simple as it it, is your first test. You need to be able to reliable assemble this dashboard before you can go onto something more interesting.

This next sample dashboard is very popular. You simple draw or import an appropriate solution architecture diagram, and then create summary alerts for each of the systems you are monitoring. A summary alert takes an arbitrary collection of alerts and displays the most urgent state. Thus, for the previous figure, App1 would show green (4/4), App2 would show red (1/4), App3 would show green (4/4), for a Summary Alert. You don't have to use all four alerts but why not? These four alerts cover everything possible state of your application: disconnected, connect, app server hung, app server active, performance problem, capacity problem, etc. With summary alerts, and the appropriate system architecture diagram, anyone can observe which systems are having problems and which experts to involve.



Building dashboards is easy, provided that you have done a correct job defining your metric groups. All the work is really in defining which metrics to focus on, how to organize them for applying thresholds, and then to assign the threshold definition. You only need to follow the right sequence of activities. And it all begins with the baseline.

SECTION 2: OPPORTUNITY

When Do You Collect Baselines?

Your first exercise of the application, while in a testing environment, is the best place to start collecting baselines. Everything you will want to do with APM: alerting, dashboards, reports and triage – these all depend on effective baselines.

Taking the time to establish a valid baseline means that no one else has to repeat that work.

If you have not yet started with baselines but the application is being monitored, then you need only use the historical view to begin identifying the components that will constitute the baseline. You will need to allow about 30 minutes to complete this task. Of course, when an urgent problem urgent problem has been detected it may seem like a luxury to complete this task. But it is essential because if you had the baseline defined, collecting a new baseline will only take a few minutes. This is obligatory, after a fix or configuration change, in order to confirm that the problem has been addressed.

Selecting a normal period in production, or a representative test during QA, takes some consideration to ensure that the period for the baseline is representative. This is discussed in the previous section: “Finding a normal period”. If you collect your baselines in advance, you simply save more time when a problem occurs.

Once the baseline is defined, it I captured in a report template. This template can then e used to generate a baseline report periodically (daily, weekly). This way you will always have a current baseline available and you will save additional time during an incident.

Different kinds of baselines

SECTION 3: BENEFITS

Effective Use of the APM Workstation

The workstation component of your APM solution is a very powerful tool. Collecting the baseline is not just something fun to do – it is a basic competency in using the workstation to do something that benefits all of your stakeholders.

Not all of your stakeholders will be able, or interested to access the APM workstation. Many will be quite satisfied with the browser-based access – provided you give them effective dashboards. Building the dashboards is easy. Knowing what to put on those dashboards – that is where the baseline will provide guidance.

Understanding the key components of your application

Every application is different but what makes your application unique? That’s what a baseline process helps you to establish. It is a mechanical process that gets you focused on the components that matter. Why they matter – that is another discussion and one that requires a lot more expertise in application architecture and software design. Generating the baseline and sharing it among your stakeholders is easy.

There are dozens of different components and thousands of metrics but only a few of them need to be exposed in order to understand and manage performance. That’s what the baseline presents – just the simple facts about what matters for your application.

More efficient triage of performance problems

When a performance problem is detected, stakeholders need facts. In a complex service implementation, every application is suspect until you can establish that you are either part of the problem, or that you are not. Comparing your performance during the incident, with your baseline, is how you make that conclusion. It lets you separate out the normal activity. Anything that remains is suspect.

Better collaboration with your stakeholders.

Knowledge is your understanding of the performance aspects of the application. You need to share this knowledge with your stakeholders for them to appreciate and value your expertise with APM.

SECTION 4: CONCLUSIONS

Baselines are the first step to understanding how to manage and communicate the performance of your application. They exercise some basic skills with the APM workstation but when you collect effective baselines, no one else needs to repeat that process. You improve collaboration by sharing this succinct information so that your stakeholders are fully informed as to what is critical about your application and where to invest to keep performance on track.

APM is how your organization comes together to manage the delivery of services. Baselines are the language that lets you track software quality and identify where components are underperforming.

SECTION 5: REFERENCES

The key reference for using the APM Workstation is the "Workstation User Guide", version 9.0.

Here are the chapters and sections that will provide additional guidance for the techniques discussed in this technical brief:

Chapter 2 – Using the Workstation Console

Viewing Historical Data

Chapter 3 – Using the Workstation Investigator

Tab Views in the Investigator's Browse Tab

Chapter 6 – Introscope Reporting

Chapter 7 – Creating and using Management Modules

Configuring Metric Groupings

Creating and Editing Dashboards

Monitoring Performance with Alerts

SECTION 6: ABOUT THE AUTHOR

Michael Sydor is an Engineering Services Architect specializing in Best Practices for APM. He advises and leads client teams to establish their own APM disciplines to deliver effective triage and manage performance across the application lifecycle. Michael is also the author of "Application Performance Management – Realizing APM", to be released later this year under the CA Press program.