APPDYNAMICS

An AppDynamics Business White Paper

SUPPORTING BLACKBOARD, KUALI, ONCORE AND MORE

4 Critical Strategies for Managing Higher Ed Apps

Application platforms like Blackboard, Kuali and OnCore have helped thousands of universities modernize their course registration, records management and even financial systems. For schools that had been using homegrown applications or archaic card systems, these applications have made lives immensely easier. However, there are inherent risks in purchasing or downloading a third party application. When something goes wrong, you must rely on your vendor's support team – or, if it's an open source solution, your own wits – to get the application back up and running before your students start complaining.

"AppDynamics lets us bridge the gap between anecdotes from users and actual, actionable information."

–Kevin Kronenbitter, *Technical Lead at Cornell University*

Universities such as Cornell University and Washington University of St. Louis have chosen to use Application Performance Management (APM) to help them troubleshoot performance problems in their third party applications on the fly. With an APM solution in place, sys admins and other IT staff at these universities are able to provide support teams with fine-grained detail about performance problems in production. When selecting an APM tool, however, these universities discovered that not all solutions are created equal. A good APM solution must be extremely intuitive, far-ranging in its capabilities, and able to speak the language of business – not the language of developers.

In particular, they selected their APM tools with the four following strategic end goals in mind:

– **SIMPLE TO INSTALL AND USE**
– **ABLE TO GAIN FULL VISIBILITY**
– **ABLE TO GAIN DEEP DIAGNOSTICS**
– **ABLE TO UNDERSTAND BUSINESS TRANSACTIONS**

Without all of these capabilities, an APM tool may be able to support certain application performance functions but will ultimately be ill-equipped for the demands of a quickly changing IT environment.

1. Simple to Install and Use

Many APM tools are difficult to install, use, and maintain. But within the APM space, there are standard practices available, such as the ability to auto-discover an application's architecture and quickly alter the mapping of the application's topology when agile release cycles introduce code changes. Your APM tool should leverage those best practices and make application management extremely easy for you.

For example, an APM tool should not require more than a series of simple steps in order to be installed. It should be up, running, and instrumenting the distributed application within hours or even minutes. The process looks something like this: the end user installs the agents on all managed servers and virtual machines, installs the controller, and re-starts the application. At that point, the tool itself should be able to handle the challenge of mapping all the databases, tiers, and nodes within the distributed application, as well as displaying those relationships in an intuitive and visual way.

However, with regards to many APM solutions, it's necessary to perform a great deal of "post-install" work—mostly aimed at tying pieces of the data flow together, manually mapping the relationship between the JVMs, setting up instrumentation for visibility, preparing the aggregation for all the data collectors, and identifying the logical tiers. Unless the APM tool is discovering this information for you, the "simple implementation" quickly turns into a mountain of post-install work.

"When they saw the call stack analysis, the programmers went 'Bingo!' and we were able to fix an issue that had plagued us for a long time."

–Bijoy George, *Biomedical Informatics Program Manager at Washington University in St. Louis*

QUESTIONS TO ASK AN APM PROVIDER:

- What is the installation process up front?
- What kind of post-install work is required?
- How does the tool map the application?
- How does it display that mapping?
- Does it keep up with the needs of agile development?
- How much manual instrumentation is needed for complete visibility?

2. Able to Gain Full Visibility

Traditional applications often consisted of little more than a few JVMs talking to one another. In the present day, a typical IT environment consists of a multitude of open source and proprietary components and distributed tiers, all attempting to communicate together in order to perform complex business transactions.

If an APM tool captures just a piece of this massive web of communication—but is unable to reveal the entire architecture, leaving blind spots in place—then the solution will fail to provide value. At the same time, the tool should not just provide visibility into what already exists, but be able to automatically discover, trace, correlate, and visualize transaction performance when agile development cycles introduce new code into the environment.

QUESTIONS TO ASK AN APM PROVIDER:

- Does your tool have insight into the entire distributed environment?
- Are there any blind spots in your current architecture?
- If it does map the environment, how does that data come back to you? Is it simple to view?

3. Able to Gain Deep Diagnostics (and That Means In Production Too)

New tools must be introduced into the production environment with extreme caution, care, and planning. That's why many APM tools aren't used in production, even if they're adept at illuminating a highly distributed application. The ability to go "wide" often precludes the ability to go "deep" in production, where adding too much overhead remains an ever-present danger.

And yet, without the ability to go deep on demand—to isolate a problem at the code level without calling in the developer who wrote the code—the APM tool won't be a reliable asset. It's necessary to drill down and see method-level detail of a particular transaction, and understand whether the incident is caused by resource constraints, bad code, or any number of potential culprits.

How can the tool retain the ability to "go deep" but still be "always on," ensuring that the diagnostic process does not result in significant overhead? Clearly, this means something other than manually instrumenting all the classes and the code paths—which by itself could still result in blind spots, and will very likely result in unacceptable overhead. Rather, the APM tool must be able to capture details of a problem whenever a problem occurs. This sounds glaringly obvious, but many APM tools work poorly in production as a result of dragging down the application with a lack of "always on" diagnostic capabilities.

This is a key point: an APM tool needs to go both deep and wide in production, being "always on" but without bringing significant overhead to bear on the application. If the tool causes more than 2% overhead, it will not be usable by anyone other than the team that developed the app.

QUESTIONS TO ASK AN APM PROVIDER:

- Can your APM tool go deep as well as wide?
- How does it isolate problem areas at the code level?
- How much overhead does it add? Less than 2%?

4. Able to Understand Business Transactions

The ability for an APM tool to focus on business transactions is important. It allows the tool to create a common language between developers and IT operations by representing the transaction, rather than a snippet of code.

A business transaction represents a "user generated" action. For example, a student might register for a class in Blackboard, or a member of your financial services department might approve a purchase order in Kuali Financial System (KFS). The APM tool needs to be able to make these actions highly visible to the IT operations team. This is an essential part of the simplicity and usability of the APM tool: the ability to talk in the language of business.

Once the APM tool gives you a clear view into the application's business transactions, then it's possible to measure the performance of those transactions. In legacy APM tools, it's necessary for you to do this by setting manual thresholds for each transaction. For example, you might say that the class registration transaction takes two seconds on average, or the student log-in takes half a second.

The problem with this approach is that you may not have the data to set those thresholds yourself. You could make an educated guess, but the burden is on you to tell the APM tool how well your application performs. If the transaction responds to load differently on Sunday versus Monday, or at 9 a.m. versus 8 p.m., or in late November versus the middle of July, it's up to you to specify the appropriate threshold for each time period. If your performance policies aren't granular enough to reflect the true performance profile that occurs during the hours in which you operate, as well as to account for periodic variations, you'll either lack visibility or be flooded with false alarms.

An APM tool that leverages best practices should be able to set those thresholds for you. This is often called dynamic baselining. This means being able to set baselines for your application by discovering how each transaction's performance may vary over specified operating periods. It observes periodic variations, accounts for them, and sets thresholds accordingly. A tool that sets dynamic baselines for each business transaction will be highly accurate and eliminate false alarms.

QUESTIONS TO ASK AN APM PROVIDER:

- How do you define "business transactions"?
- How does your tool represent them?
- How does your tool speak the "business language" of the application?
- Does the tool learn and react from performance baselines dynamically? If not, how are they set and maintained?

Summary

Managing the performance of your Blackboard, Kuali or other higher ed application requires a strategic approach, particularly in a space crowded with multiple application performance solutions boasting very similar messages. To assist your decision, leverage these four strategies for application performance management—and ensure the ongoing health and reliability of your revenue-critical applications.

Try it FREE at
www.appdynamics.com