APPDYNAMICS

AppDynamics Database Visibility

Evaluators guide

Table of contents

Table of contents

Slow database queries ranked as the #1 most common root cause of application performance problems.

– The Dzone Guide To Performance and Monitoring 2015 Edition

Introduction

This document is designed for anyone with a stakeholder interest in the health of databases within the context of application performance. This list may include:

- Database administrators
- Production support
- Developers
- Performance testers
- Anyone whose phone rings when the application is having problems!

What is AppDynamics for Database?

AppDynamics Database Visibility delivers agentless heterogeneous database monitoring capabilities in the context of applications that they serve. This deep visibility can help companies not only quickly identify and resolve any database performance bottlenecks but also understand its impact on the application and end-user experience.

How does AppDynamics Database Visibility work?

AppDynamics continuously monitors test or production database instances 24x7 and stores detailed historical data about the resource consumption and wait events in a repository. This data is correlated, aggregated and sorted so that it can be easily displayed in the intuitive web-based user interface.

This historical data allows the DBA to answer questions such as: “What caused the slowdown in the application yesterday after lunch?” or “Why is the overnight batch job still running at 9 am this morning?” The historical data can also be used in a proactive manner; for example, to assess upward trends in SQL response time or resource consumption or to tune and fix issues before they become end-user problems.

Supported Databases

- Oracle 10g, 11g & 12c
- SQL Server 2005 – 2014
- SAP/Sybase ASE 15.0-16.X
- MySQL 4+ (including MariaDB and Percona)
- SAP/Sybase IQ
- MongoDB 2.X+
- IBM DB2 LUW 9.5+
- PostgreSQL 8+
- Microsoft Azure SQL

All Hardware/OS Supported

Supported Monitored Operating Systems

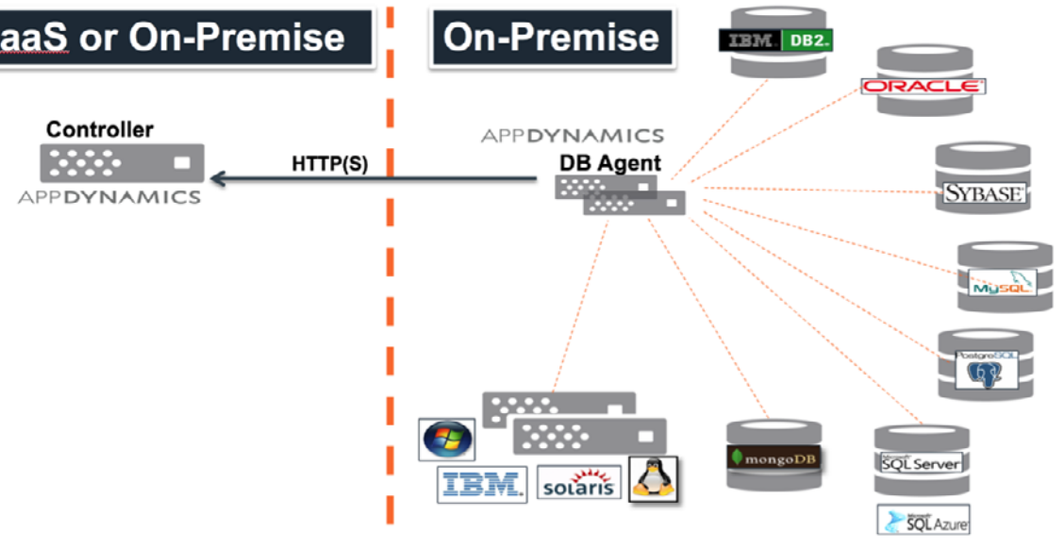
- Windows
- Linux
- Solaris
- AIX

Client Support

- Internet Explorer, Firefox, Chrome, Safari

Agentless technology

AppDynamics Database Visibility does not require any software agents on your monitored database servers; instead, agentless technology is used to monitor from a remote central server, which negates deployment issues and monitoring overhead. AppDynamics is ideally suited to cloud, virtual or physical architectures and performs great in high-load production environments.



“The best thing about AppDynamics for Databases in production is the amount of time it saves us when investigating performance problems. This means we fix problems faster and keep our customers happy. It automated our approach to performance tuning and removed the need for manually reviewing data from different tools.”

– Unai Basterretxea, DBA Engineering Manager
betfair

Why you need AppDynamics Database Visibility

Business impact

AppDynamics can be installed and monitoring within minutes, which means that the information can be used right away to solve real business issues and deliver immediate value, so you gain:

- Reduced MTTR
- Reduced debate among teams
- Increase Application Performance
- Increase Brand Loyalty

Complete the APM picture

AppDynamics Database Visibility can monitor any combination of supported database platforms from a single central installation, which means that database performance management can be standardized using a best-of-breed performance monitoring methodology. Software is inclusive of the Database, so why only monitor software and not the DB? With a unified view, you can:

- Correlate bad performance calls from the application to the offending database.
- Context and actionable information, not just DB metrics.
- End-to-End Visibility, all in the context of a business transaction

Accurate diagnostics

Answer questions such as: “What caused the slowdown in the application yesterday after lunch?” or “Why is the overnight batch job still running at 9 am this morning?” This historical data can also be used in a proactive manner; for example, to assess upward trends in SQL response time or resource consumption or to tune and fix issues before they become end-user problems. This will result in:

- No more finger pointing
- Understanding beyond simply what Database call is bad and where, to also why it is bad and how to solve it.
- Insight into popular Database performance issues, such as:
- Bad Query Performance
- Bad Indexes
- Users/Queries Conflicts
- Capacity Issues
- Inefficient Configuration Parameters

Unite teams on DB issues

Now with a single pane of glass for all your AppDynamics products, you can achieve faster problem resolution from fully integrated data, correlation, and analytics. Who is it for?

- Database administrators
- Production support
- Developers
- Performance testers

No need for separate infrastructure since AppDynamics Database Visibility does not require any software agents on your monitored database servers. Thus, you can focus on solving:

- Disconnect among teams
- No more data silo between database performance diagnostics

Know the “Why” with actionable information

As AppDynamics database instances, it stores detailed historical data about the resource consumption and wait events in a repository. With this historical data, you can diagnose problems such as:

- Slow database response times
- Database load issues
- Unpredictable performance spikes
- Locking problems
- Internal database contention

Did you know that databases are the #2 reason for application performance problems?

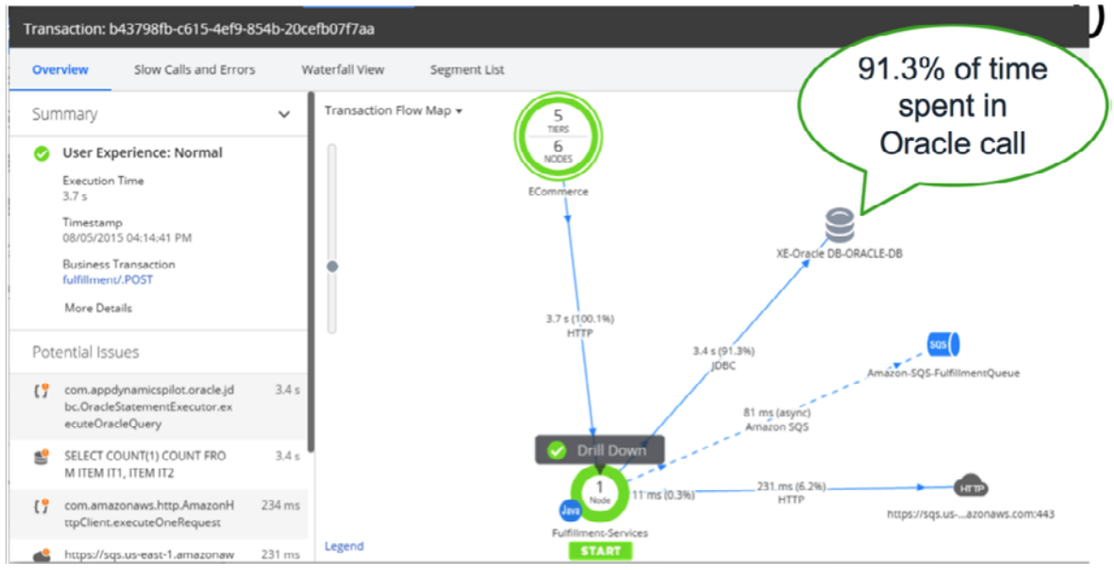
– The Dzone Guide To Performance and Monitoring 2015 Edition

Six database monitoring use cases

Query performance

The most obvious place to look for poor query performance is in the query itself. Problems can result from queries that take too long to identify the required data or bring the data back. Look for these issues in queries:

- Selecting more data than needed
- Inefficient joins between tables
- Too few or too many indexes
- Too much literal SQL causing parse contention



Bad indexes

If the number of write functions to a table and its indexes is exceedingly greater than the number of reads to it, there is a good chance the underlying indexes are harming overall performance.

Every time a write is performed to an SQL column that has an index, a corresponding modification must also be activated to the column indexes.

If a lot of the activity is write activity, it might be worth it to consider removing or altering the indexes involved. Doing so would likely increase performance by reducing the overall output of write activity. Find bad indexes by using Dynamic Management Views to analyze query execution statistics. After finding the indexes that have many writes but zero or few reads, consider dropping those indexes to improve performance.

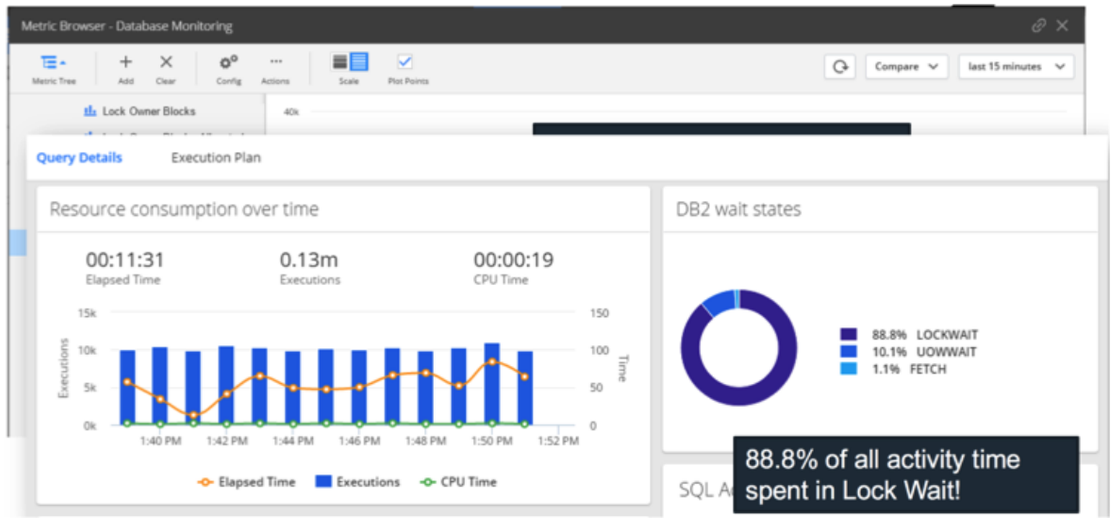
When there aren't any indexes that the query optimizer can use, the database needs to resort to table scans to produce query results, which generates a large amount of disk input/output (I/O). Proper indexes also reduce the need for sorting results. Indexes on non-unique values do not provide as much help as unique indexes in generating results. If the keys are large, the indexes become large as well, and using them creates more disk I/O. Most indexes are intended to help the performance of data retrieval, but it is important to realize that indexes also impact the performance of data inserts and updates, as all associated indexes must be updated.

Users/queries conflicts

Databases are designed to be multi-user, but the activities of multiple users can cause conflicts.

Page/row locking due to slow queries

To ensure that queries produce accurate results, databases must lock tables to prevent inserts and updates from occurring while a read query is running. If a report or query is slow, users who need to modify database values may experience slowness and delays in completing their updates. Lock hints help the database use the least disruptive locks. Separating reporting from transactional databases is also an efficient solution.



Transactional locks and deadlocks

Deadlocks occur when two transactions are blocked because each one needs a resource held by the other. When there's a normal lock, a transaction is blocked until a resource is released. There isn't a resolution to a deadlock. Databases monitor for deadlocks and choose to terminate one of the blocked transactions, freeing the resource and allowing the other transaction to proceed. The other transaction is rolled back.

Batch activities causing resource contention for online users

Batch processes typically perform bulk operations such as loads of large amounts of data or generating complex analytical reports. These operations are resource-intensive and can impact performance for online users. The best solution for this issue is to ensure that batch operations are run when online usage is low, such as at night, or to use separate databases for transactional processing and analytical reporting.

Capacity issues

Not all database performance issues are database issues. Some problems result from running the database on inadequate hardware.

Not enough CPUs or CPU speed too slow

More CPUs can share the server workload, resulting in improved performance. The performance the database experiences is not solely due to the database but also is affected by other processes running on the server, so it is important to review the overall load as well as database usage. As CPU utilization varies throughout the day, metrics should be examined for periods of low usage, average usage, and peak usage to best assess whether additional CPU resources will be beneficial.

Slow disk without enough IOPS

Disk performance can be stated in terms of input/output operations per second (IOPS). Combined with the I/O size, this provides a measure of the number of the disk's throughput in terms of megabytes per second. This throughput is also affected by the disk's latency, which is how long it takes the request to complete. These metrics are unique to the technology of the disk storage. Traditional hard disk drives (HDD) have a rotating disk and are typically slower than solid state drives (SSD) or flash memory without any moving parts. Until recently, an SSD was more expensive than an HDD, but costs have come down, making it a competitive option.

Full or misconfigured disks

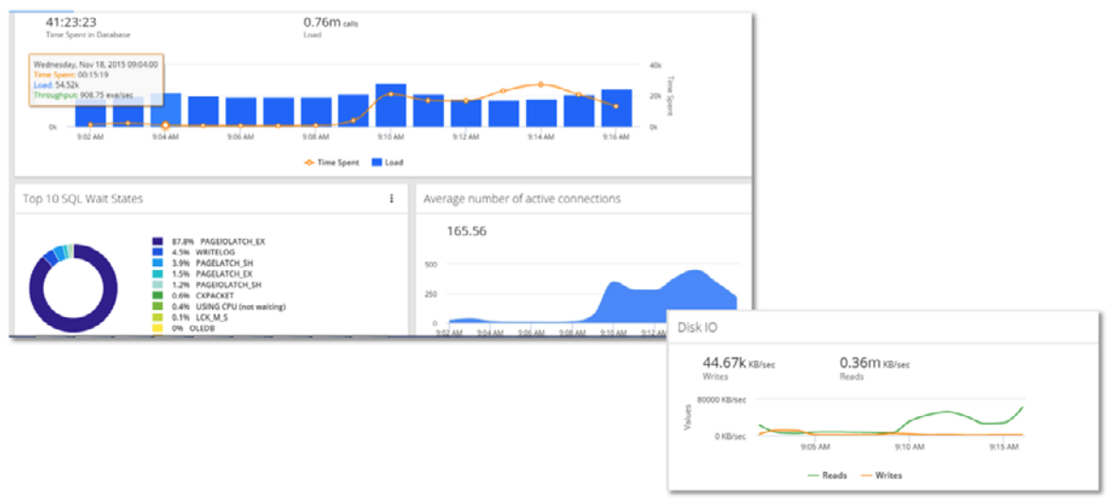
Databases obviously require significant disk access, so incorrectly configured disks have a considerable performance impact. Disks should be suitably partitioned, with system data such as catalogs and logs separated from user data. Highly active tables should be separated to avoid contention. Increase parallelism by placing databases and indexes on different disks. Don't arrange the operating system and swap space on the same disk as the database.

Not enough memory

Limited or poorly allocated physical memory impacts database performance. The more memory that is available, typically the better the performance will be. Monitor paging and swapping. Set up several page spaces on multiple, non-busy disks. Make sure the paging space allocated is sufficient for database requirements; each database vendor can provide guidance on this matter.

Slow network

Network speeds can affect how quickly retrieved data is returned to the end user or calling process. Use broadband for connecting to remote databases. In some cases, choosing TCP/IP instead of named pipes for the connection protocol can significantly increase performance.



Inefficient configuration parameters

Every database has a large number of configuration settings. Default values may not be enough to give your database the performance it needs. Check all parameter settings, which includes looking for the following issues:

Buffer cache too small

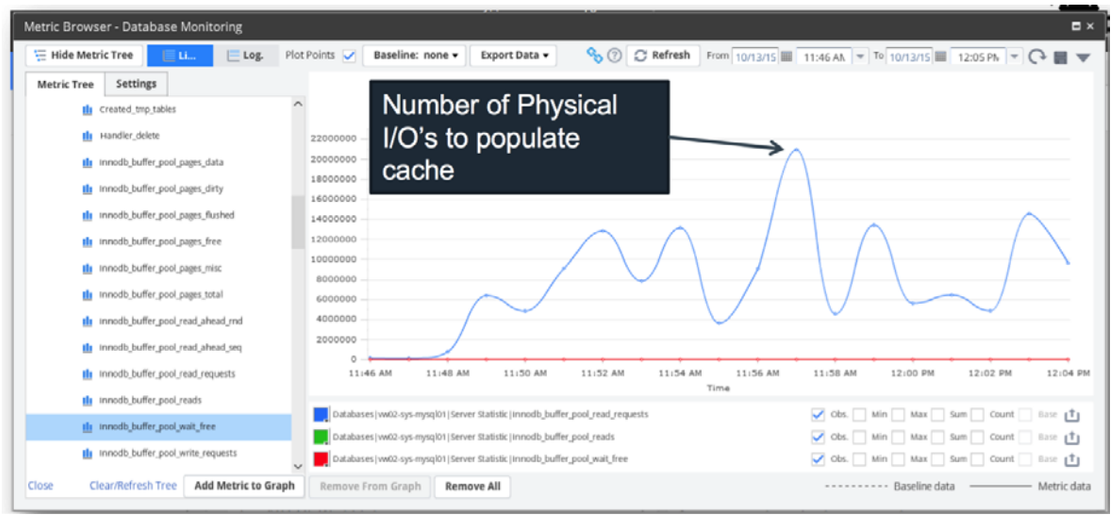
Buffer cache improves performance by storing data in kernel memory and eliminating disk I/O. When the cache is too small, data is flushed from the cache more frequently. If it is needed again, it must be reread from disk. Besides the slowness of the disk read, this puts additional work on I/O devices and can become a bottleneck. In addition to allocating enough space to the buffer cache, tuning SQL queries can help them use buffer cache more efficiently.

No query caching

Query caching stores both database queries and their result sets. When an identical query is executed, the data is quickly retrieved from memory rather than requiring the query to be executed again. Updates to data invalidate the results, so query caching is only effective on static data. In some cases, a query cache can become a bottleneck rather than a benefit to performance. Huge caches can cause contention when they are locked for updates.

I/O contention due to temporary table creation on disk

Databases need to create temporary tables when performing certain query operations, such as executing a GROUP BY clause. When possible, the temporary tables are created in memory. However, in some cases, creating the temporary table in memory is not feasible, such as when the data contains BLOB or TEXT objects. In those cases, the temporary tables are created on disk. A large amount of disk I/O is required to create the temporary table, populate it with records, select the needed data from it, and drop the table when the query is complete. To avoid potential performance impact, the temporary database should be separated from the main database space. Rewriting queries also reduce the need for temporary tables by creating derived tables instead. Using a derived table, which directly selects from the result of another SELECT statement, allows data to be joined in memory rather than using disk.



NoSQL databases

NoSQL has much appeal because of its ability to handle large amounts of data very rapidly. However, some disadvantages should be assessed when weighing if NoSQL is right for your use-case scenario. This is why it is wise to consider that NoSQL stands for "Not Only SQL." This clearer definition accepts the premise that NoSQL is not always the right solution, nor does it necessarily replace SQL across the board — here are five reasons why:

Finicky transactions

It is hard to keep entries consistent with NoSQL. When accessing structured data, it does not always ensure that changes to various tables are made at the same time. If a process crashes, tables might become inconsistent. An example of consistent transactions is double-entry accounting. A corresponding credit must balance every debit and vice versa. If the data on both sides is not consistent, the entry cannot be made. NoSQL may not "balance the books" properly.

Complex databases

Supporters of NoSQL tend to point to the efficient code, simplicity, and speed of NoSQL. All of these factors line up when database tasks are simple. However, when databases become more complicated, NoSQL begins to break down. SQL has more potential over NoSQL when database demands are complicated because SQL has mature, industry standard interfaces. Each NoSQL setup has a unique interface.

Consistent JOINS

When executing a JOIN in SQL, there is a tremendous amount of overhead because the system must pull data from different tables and align them with keys. NoSQL seems like a dream because there is a lack of JOINS. Everything is all together in one place in the same table. When data is retrieved, it pulls all of the key-value pairs at the same time. The problem is that this can create several copies of the same data. Those copies have to be updated, and NoSQL does not have the functionality to help in this situation.

Flexibility in schema design

NoSQL was unique when it emerged on the scene because it did not require a schema. In previous database models, programmers would have to think about the columns they needed to accommodate all of the potential and data entries in each row. With NoSQL, entries can have a variety of strings or none at all. This flexibility allows programmers to ramp up applications quickly. However, it can be problematic when there are several groups working on the same program, or when new teams of developers take over a project. After some developers have modified the database using the freedom of NoSQL, there may be a wide variety of key pair implementations.

Resource intensive

NoSQL databases are commonly much more resource intensive than relational databases. They require much more CPU reserves and RAM allocation. For that reason, most shared hosting companies do not offer NoSQL. You must sign up for a VPS or run your own dedicated server. On the other hand, SQL is made to run on one server. This works out fine in the beginning, but as database demands increase, the hardware must expand as well. The problem is that a single server with huge capacity is much more expensive than a variety of smaller servers. The price increase is exponential. This provides one reason NoSQL has found a home in enterprise computing scenarios, such as those used by Google and Facebook.

“AppDynamics for Databases helped us immensely, and as a result we achieved the performance loads required within the limited time available before go-live. It was definitely money well spent. We have no hesitation in recommending AppDynamics for Databases.”

– Jag Patel, Development Manager
British Heart Foundation

Glossary

Wait/queue state

A period of waiting that comes after executing queries or loading resources related to particular tasks. While SQL is executing one or more queries or pulling resources, a certain amount of time must be spent both scanning the storage and data and performing the calculation or task at hand.

Latch wait: Refers to a wait that happens when a task is waiting on a latch for an I/O request type of buffer (e.g. PAGEIOLATCH_EX)

CXPACKET wait: a typical problem related to high server CPU usage due to poorly-written parallel queries (queries designed to run concurrently).

WRITELOG wait: related to the SQL session writing the contents of the cache of a log to the disk where the log is stored.

Locking

Lock resources: the places where SQL can place locks.

Lock modes: the locks that can be placed on resources so they can be accessed by concurrent tasks and transactions.

There are several resources where locks can be placed, such as a row in a table or a lock on each row within an index. There are also several lock mode types, such as shared locks and exclusive locks. Some locks are completely fine, but others can be detrimental to performance.

Disk & Network I/O

SQL data and transactions funneling in and out of the disk, cache or through the network. The more there are, the worse the performance can be. However, fine-tuning your queries and indexing can significantly reduce the input and output on the physical and logical disks and network.

Contention

Contention can occur, for example, when processes are trying to perform updates concurrently on the same page. Typically a term related to contention in locking. Locking in SQL helps to ensure consistency when performing read or write tasks in the database, but contention when locking can happen.

High CPU Usage

High server CPU usage as it relates to SQL is directly connected to:

- the SQL processes being run
- poor query execution
- system tasks and excessive compilation and recompilation of queries.

The CPU can also be strained if there are bad indexes in place.

AppDynamics

AppDynamics APM is the next generation application performance management solution that simplifies the management of complex, business-critical apps. No one can stand slow applications – not IT Ops and Dev teams, not the CIO, and definitely not end users. With AppDynamics APM, no one has to tolerate slow performing apps ever again. We allow companies to monitor, troubleshoot, diagnose, and scale their production applications – gaining 10x their current level of visibility and getting to root cause 90% faster.

Distributed web applications often contain frustrating blind spots and mysterious, recurring problems. Only AppDynamics APM delivers the simplicity, visibility, and deep diagnostics that Ops and Dev teams require.

The new world of distributed web applications has created a whole new set of challenges for those tasked with ensuring application health and performance. Modern application architectures, new technologies, and a rapid rate of change have created a perfect storm of complexity in today's applications. As a result, performance problems surface that are often difficult to identify, diagnose, and fix.

As these applications become increasingly critical to the business, it's more important than ever to have a simple yet fast way to monitor, diagnose, and resolve application problems before they affect revenue.

Introducing AppDynamics APM

The AppDynamics Application Performance Management solution provides business transaction-centric management of distributed applications. The solution is extremely easy to configure and deploy, consumes little production overhead, monitors every line of code, and baselines performance to proactively identify and resolve application performance issues before they impact customers and the business.

What Makes AppDynamics Different?

- **Auto-discover and monitor end-to-end business transaction performance, with Transaction Tag and Follow**
  - Automatically discover application topology and interdependencies including external web services, and trace key business transactions based on production application behavior
  - Visualize and prioritize the business transactions performance and not just the health of the application and infrastructure nodes
- **See everything with the industry's broadest coverage of languages and technologies**
  - Covers all popular programming languages and frameworks including Java, .NET, Node.js, PHP, Python and C/C++ (Beta)
  - Covers most complex enterprise platforms and solutions such as JMS, queuing technologies, TIBCO, WebMethods, etc.
  - Leverage platform extensibility for wider application monitoring coverage
- **Monitor any production app within minutes**
  - Zero configuration to monitor any app
  - Monitor the app even when you don't have its source code
- **Monitor production apps at code-level depth with no overhead**
  - Leverage Smart Code Instrumentation to enable in-depth monitoring of production apps without making configuration changes
  - Monitor every transaction but intelligently capture details of only the anomalous transactions, making the platform scale to meet the demands of large enterprises

Learn more at appdynamics.com

APPDYNAMICS

appdynamics.com