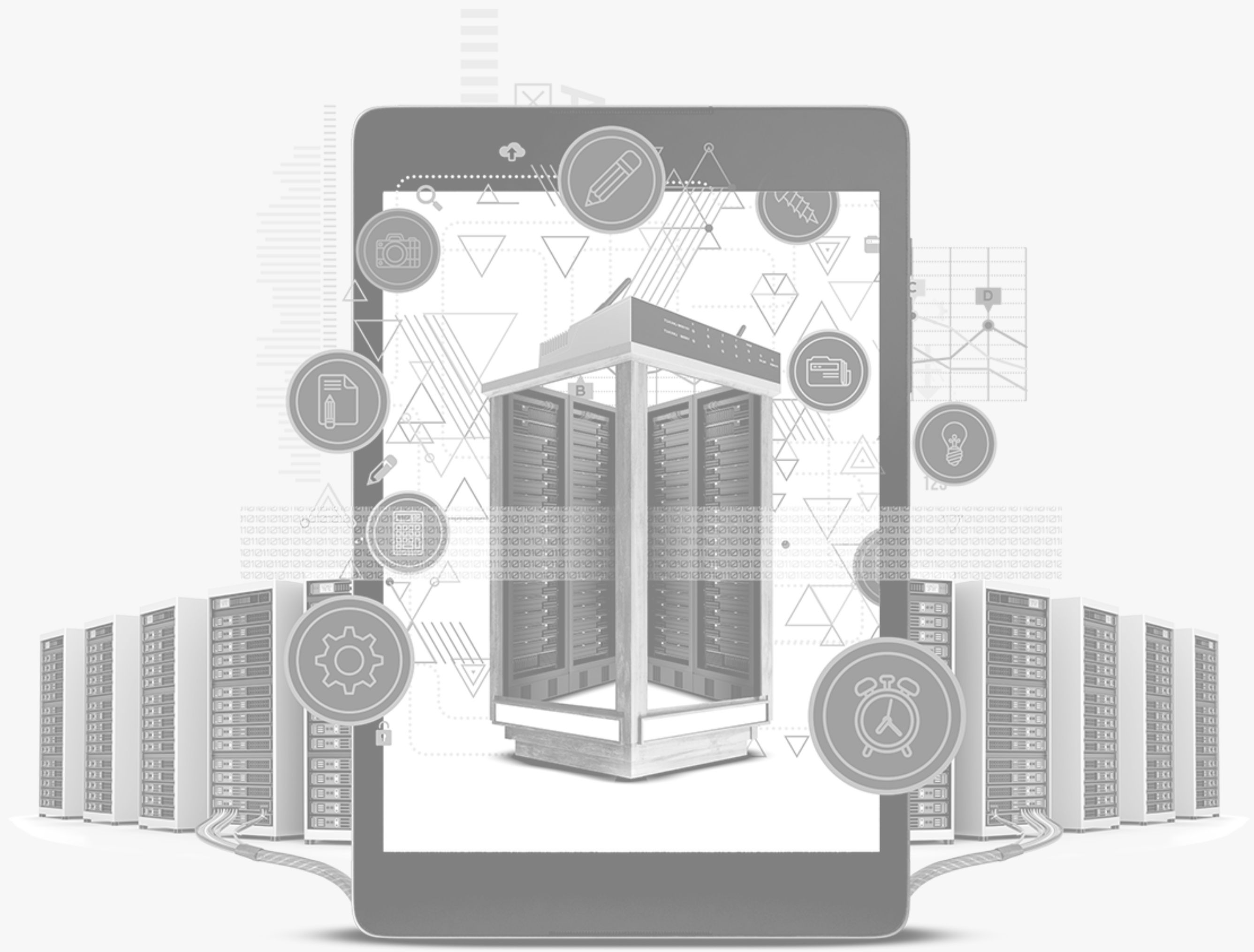
solarwinds



eBOOK

APM Back to Basics

Table of Contents

Abstract

Managing application performance can be a daunting task and the marketing buzz surrounding different application performance management (APM) tools can be confusing. This eBook reviews the core features of APM tools, dives deep into what they mean (and in some cases, how they are implemented), and discusses the monetary costs and ROI of an APM tool. After reading this eBook, you should better understand what APM is and how to choose the best tool for your application and budgetary needs.

What is APM?

Application performance management—sometimes referred to as application performance monitoring—refers to monitoring tools that allow IT, developers, and business leaders to monitor their application architecture to resolve performance issues and bottlenecks in a timely manner.

Put simply, APM is a technology that acts like an MRI machine. It allows you to view and fix parts of your application you wouldn't have been able to (or would have struggled with) without it. However, getting to the bottom of what different APM tools provide is difficult: some implementations are complex to begin with, and there are a lot of vendor marketing dollars at work to make buzzwords better align with specific approaches. This eBook reviews the core features of APM tools, dives deep into what they mean (and in some cases, how they are implemented), and discusses the monetary costs and ROI of an APM tool. Each vendor has their own approach and unique add-ons, but you want to ensure that you are paying for what you really need, not a lot of bells and whistles that won't help you accomplish your primary objective: ensuring the performance of your applications.

Technology analyst firm, Gartner®, breaks APM capabilities into three main segments:

» 1. Digital experience monitoring (DEM)

» 2. Application discovery, tracing and diagnostics (ADTD)

» 3. Application analytics (AA)

DIGITAL EXPERIENCE MONITORING (DEM)

DEM is a monitoring strategy used to quantify and to optimize applications and digital touchpoints to provide a seamless user experience. Gartner defines DEM as follows:

"Digital experience monitoring is an availability and performance monitoring discipline that supports the optimization of the operational experience and behavior of a digital agent, human or machine, as it interacts with enterprise applications and services. For the purposes of this evaluation, it includes real-user monitoring (RUM) and synthetic transaction monitoring (STM) for both web- and mobile-based end users."

APPLICATION DISCOVERY, TRACING, AND DIAGNOSTICS

This area is where nearly every APM tool provides the most value. Can your APM automatically discover your application topology and map services? Can the tool trace user requests as they traverse through your application? And finally, does it provide analysis and the ability to deep dive when there's an issue?

For the vast majority of APM purchases, it's this category that drives a purchase decision.

APPLICATION ANALYTICS

Application analytics is the ability of APM products to assist in drilling down and providing root cause analysis. When you have a performance issue with your application, application analytics helps you understand why the issue occurred, resolve it, and prevent it from happening again.

Key Features and Use Cases

There is a core set of features that all APM products must provide in order to be classified as APM products. In this chapter, we will review what they are and why they are important. In subsequent chapters, we'll dive into each one.

This set of key capabilities includes the following:

» Application Key Performance Indicator (KPI) metrics

» Infrastructure KPI metrics

» Distributed tracing

» Temporal event management

» Intelligent alerting

» Custom dashboards

APPLICATION KPI METRICS

First, application time-series metrics track things like requests per minute, average response time, and error rates over time. We say that these are "time-series" because we want to capture them at different instances of time, plot them over time, and identify changes or trends in their behavior. They can be analyzed and presented at different scopes, including at an application level, service level, transaction level, as well across calls to external dependencies.

Time-series metrics are important for APM products because they can identify performance issues as they occur, as well as trends in performance data before performance issues affect your users. Application time-series KPI metrics are at the core of the value that APM products provide.

INFRASTRUCTURE KPI METRICS

Next, infrastructure KPI metrics measure things like CPU utilization, memory utilization, and disk and network I/O. These metrics are important to capture because they characterize the behavior of the infrastructure on which your applications run. If an application is not yet experiencing performance issues, but its underlying infrastructure is trending problematically, these metrics can provide an early warning to help you remediate the problem. Furthermore, infrastructure KPI metrics allow you to better distinguish between application problems and systemic problems.

DISTRIBUTED TRACING

Once you have identified a performance problem with a web or service request, the next step is to identify the root cause of the issue, which is where distributed tracing comes in. Distributed tracing allows you to view the components that are contributing to your application response time. It stitches together a request across application tiers and external dependencies, and then dissects the performance in each tier so you can see how every component in your application is contributing to your overall response time.

Distributed tracing allows you to break down an application response time and see where the majority of that time is being spent. Without this capability, an APM tool would only tell you about a problem, not where the problem is or where to start fixing it. Finally, distributed tracing (as well as all of your performance management solution) must be low overhead: the worst thing an APM tool can do is provide rich information but at the cost of slowing down your application.

TEMPORAL EVENT MANAGEMENT

One constant in IT is that nothing is constant: you can always count on change and need a way to tell your APM tool about temporal events, such as the deployment of a new release or a planned outage. Associating temporal events with performance data allows you to distinguish between application performance or outage issues and the expected behavior during the event. Furthermore, it allows your monitoring to potentially modify its behavior during the event and can help it avoid raising false alerts.

INTELLIGENT ALERTING

Next, an APM tool needs to have intelligent alerting. Alerting can include things like email notifications, dashboard visualizations, and even integrations with other systems to open a service ticket and assign it to the correct party. But in order for alerts to be intelligent, the APM tool needs to be configurable to understand the nature of your application and its behavior. This allows it to detect anomalies. It the early days of APM, it was quite common to define static thresholds, such as "raise an alert if this service call takes more than three seconds to return." As APM solutions evolved, alerting became smart enough to perform rudimentary statistical analysis and allowed for alerts based on factors like standard deviations or percentage differences from the mean.

CUSTOM DASHBOARDS

Finally, APM tools need to allow you to create custom dashboards. Many APM tools come with a collection of beautiful visualizations, but the important thing for you is a visualization of your specific business processes. You need the flexibility to create dashboards that show what's important to your business. For example, you may have a web service that handles your payment transactions, so it might be important to your business to show metrics like the rates that transactions are completing, the average amounts of those transactions, as well as the response time and error rates. APM tools have no intrinsic way to understand that a service is vitally important to your business, so it is important that your APM tool allows you to focus on important business processes via custom dashboards.

The next chapters review each of these key features in detail and provide examples that you should look for when selecting an APM tool.

Time-Series KPI Metrics

Application time-series metrics, as their name implies, are metrics captured over a period of time for the purposes of detecting performance anomalies and forecasting. Time-series metrics are not typically used for diagnosing the root cause of performance problems, but rather for detecting performance problems. Examples of time-series metrics include:

» Requests per minute

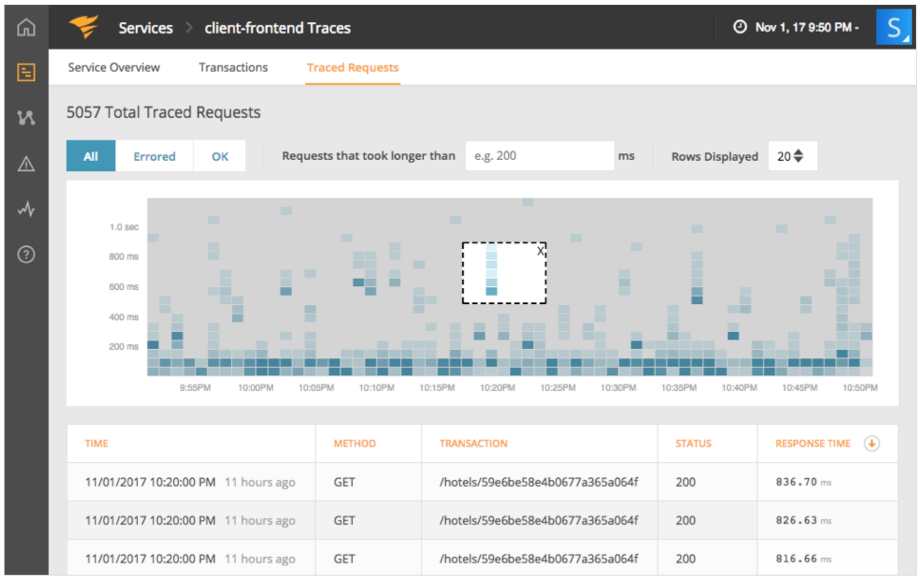
» Average response time

» Error rates

Time-series metrics allow you to view the performance of your application over time, which means that they can feed a rules engine so that it can raise alerts when things are behaving abnormally. For example, if you were to capture the average response time of key web service calls over a period of a couple days or even a few weeks, you would be able to compare the current response time of those web service calls to their historical response times and raise an alert if the current response time is, for example, more than two standard deviations from the historic mean. These metrics allow you to understand what is normal so that you can detect what is abnormal.

Because time series metrics capture historic behavior, they can also provide great insight into the future, meaning that they can be used to identify trends and drive your forecasts. You may choose to plot a simple linear regression line, or leverage a more sophisticated data science algorithm, to predict when you are going to need to purchase more hardware or scale up what you already have.

Figure 1 shows an example of time-series response time data plotted over time.



5057 Total Traced Requests

Requests that took longer than e.g. 200 ms

Rows Displayed 20

TIME	METHOD	TRANSACTION	STATUS	RESPONSE TIME
11/01/2017 10:20:00 PM 11 hours ago	GET	/hotels/59e6be58e4b0677a365a064f	200	836.70 ms
11/01/2017 10:20:00 PM 11 hours ago	GET	/hotels/59e6be58e4b0677a365a064f	200	826.63 ms
11/01/2017 10:20:00 PM 11 hours ago	GET	/hotels/59e6be58e4b0677a365a064f	200	816.66 ms

Figure 1. Time-series data

In Figure 1, we can see the variability in response time over a two-day period with a daily upward trend, although you need far more than two days of data for accurate trending. This granularity of data allows you to examine current service requests against a historic average or even by looking at the performance of this application based on the hour of day. Longer time-series data would enable you to look at response times against specific dates.

Additional time-series metrics are captured and aggregated at various granularities, including:

» Service-level

» Application-level

» Transaction-level

» External dependency

Modern enterprise and web applications are not constructed as large, standalone monoliths, but rather, are composed of many loosely coupled components, most typically services. It is important to be able to view the performance of your services individually, such as authentication and authorization services that are used across applications in your ecosystem. Degradation of an individual service's performance may have cascading effects across multiple applications, so viewing the performance individually is important.

Although modern applications are composed of loosely coupled services, they still ultimately contribute to the behavior and performance of specific business applications. As such, your APM tool needs to be able to aggregate service response times that are part of an application to that application. This provides you, and your application business owners, with the level of granularity necessary for tracking and ensuring the performance of business applications.

Applications are composed of a collection of different transactions. For example, a user may be able to log in to an application, add items to a shopping cart, and then check out. Each of these application flows represents different application transactions. While it is important to be able to understand an application's holistic performance, it is also important to understand the performance of its constituent transactions. As such, your APM tool needs to be able to aggregate performance metrics at a transaction-specific level.

Finally, applications typically interact with external dependencies by making outbound calls to systems like databases, messaging systems, caching servers, and external web services. These are not system that you typically build, but they do contribute to the performance of your applications and are important to track and to isolate from the rest of your applications. Your APM tool needs to be able to identify these external dependencies and aggregate performance metrics for calls to these external dependencies because resolving issues in external dependencies is far different from resolving issues in your application itself.

Figure 2 shows a breakdown of performance across different layers in an application, including several different databases.

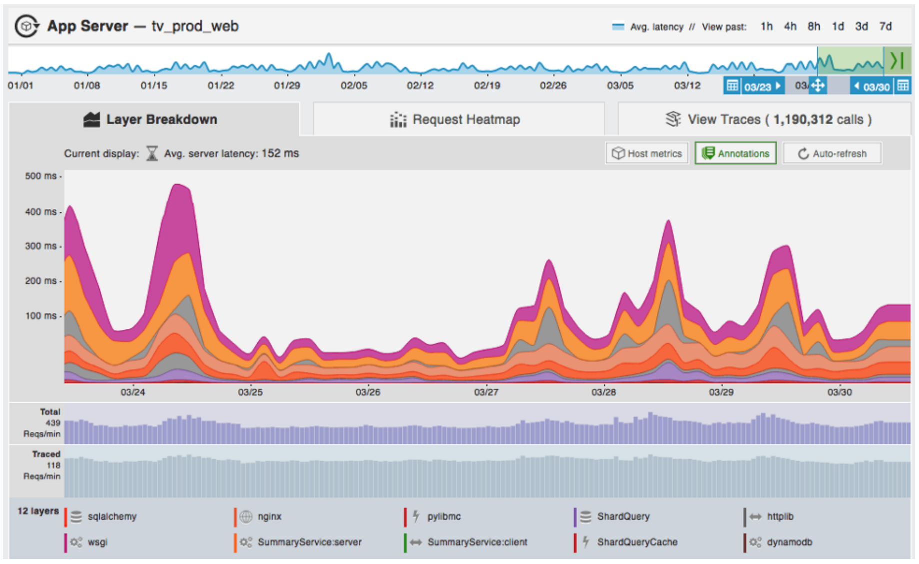


Figure 2. Time-series data broken down by layers

In Figure 2, we see the response time breakdown not only for the application code, but for each layer in the application, including external calls to destinations such as SQLAlchemy and DynamoDB®.

Time-series metrics are at the heart of APM solutions. Without these metrics, an APM tool cannot do its job. Time-series metrics provide the data required for an APM tool to detect abnormalities, raise alerts, and capture traces for detailed analysis. Additionally, they provide the means for you to perform trend analysis and forecasting.

Infrastructure KPI Metrics

Regardless of the technology you chose to build your application on, it will ultimately run on infrastructure, whether that is a container/platform as a service (PaaS) technology, virtualization technology, or directly on hardware. And the performance of your infrastructure can greatly affect the performance of your application. Figure 3 shows an example of the types of infrastructure that might be in your technology stack.

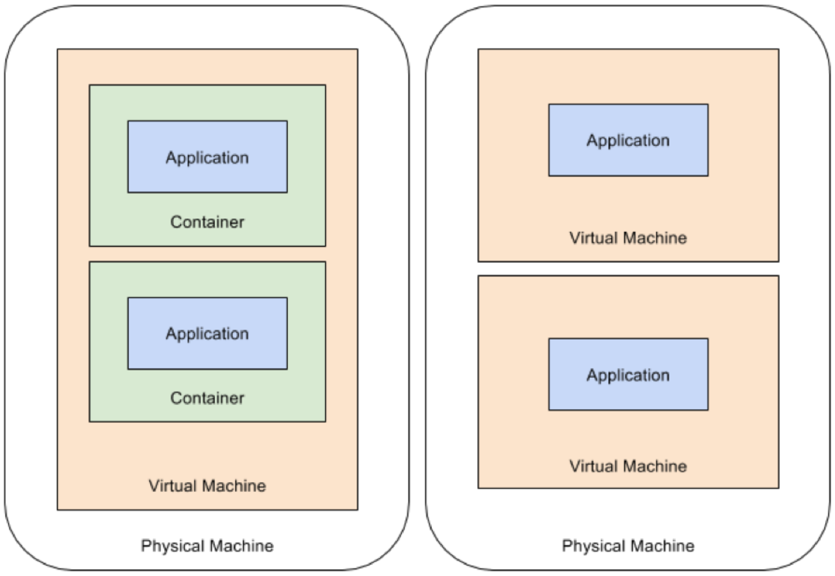


Figure 3. Infrastructure

Figure 3 is a simplified representation of a runtime infrastructure. On the left, we see applications running in in containers that, in turn, are running on a virtual machine that is running on a physical machine. On the right, we see applications running directly on virtual machines that are running on physical machines.

In order to properly determine the performance of this infrastructure, you need visibility into the behavior of the physical machine, including metrics like CPU utilization, memory utilization, disk I/O, and network I/O. You also need visibility into the behavior of the virtual machine, including metrics like vCPU utilization, vMemory utilization, and so forth. If you are running in a container, you need visibility into its behavior. And finally, you need visibility into your application technology stack.

Let's consider an example of a Java® service running in Tomcat® on a Docker® container. Figure 4 shows the types of metrics you might want to capture in this scenario.

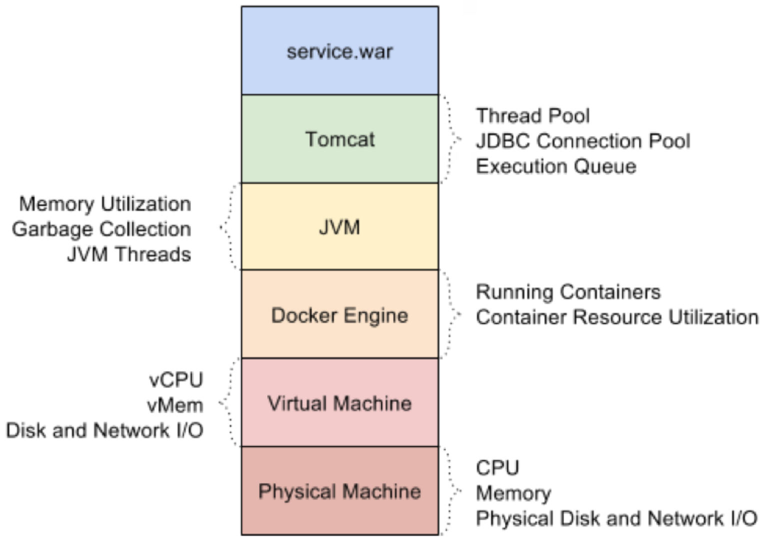


Figure 4. Java Web Service Running in Tomcat on Docker

We might consider the "infrastructure" to be everything below the actual service code (service.war in the figure above). Figure 4 shows the types of metrics that you will want to capture at every step of the in the technology stack:

» Tomcat - You need to be able to determine the health of Tomcat, which includes things like thread pools and execution queues, as well as connection pools and caches.

» JVM - Because Tomcat runs on a JVM, you need to determine the health of the JVM by looking at things like memory utilization, garbage collection, and JVM total thread usage.

» Docker - The Docker Engine is a small process that abstracts the underlying hardware from its containers. You need to capture things like the number of running containers, container logs, resource utilization, and so forth.

» Virtual Machine - The virtual machine is hosting your Docker Engine and allows it to function by using its virtual CPU, memory, disk, and network I/O.

» Physical Machine - If you are running on-premises then you will be managing physical machines that are hosting your virtualized environment. The physical machine hosts the real CPU, memory, attached and internal storage, and network card(s).

Your APM tool needs to be able to capture all of these metrics and then interpret them in a meaningful way. For example, 40% CPU utilization is actually not a good thing. You would rather see your physical machines running somewhere between 75% and 90%, because below 75% and you are wasting resources, but above 90% and you are probably starting to run into thread and CPU contention. Your tool needs to be able to interpret these types of metrics as such.

Figure 5 shows an example of the types of metrics you'll want to capture and visualize.

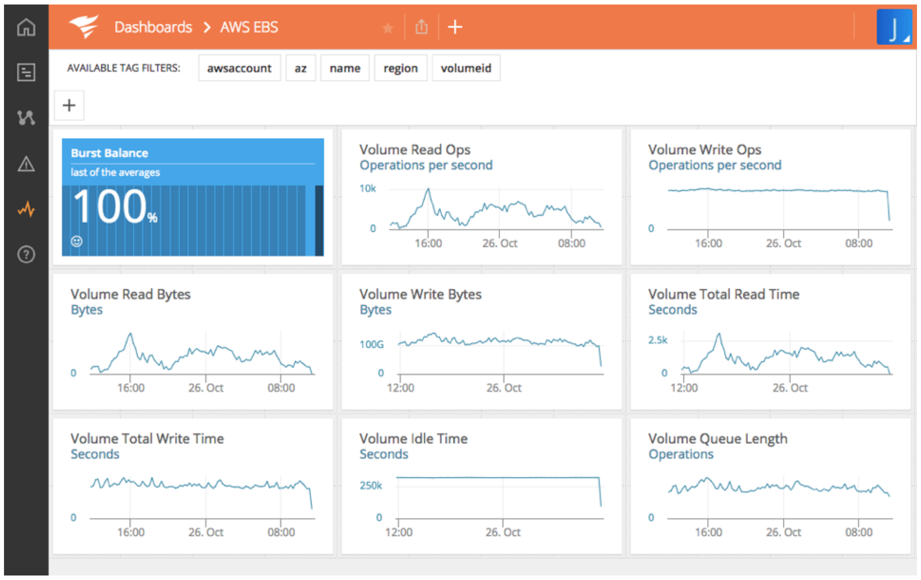


Figure 5. Infrastructure Metrics

In this example, we see a virtual machine running in AWS®, so the APM tool captured CPU utilization as well as disk and network I/O metrics.

Distributed Tracing

Time-series KPI metrics are used to detect performance issues but distributing tracing enables you to identify the root cause of issues. A trace shows both the interactions of a request as it passes between servers in your technology stack, as well as a method-level view of what is happening in each server.

Figure 6 shows an example of a simple technology stack for a distributed application.

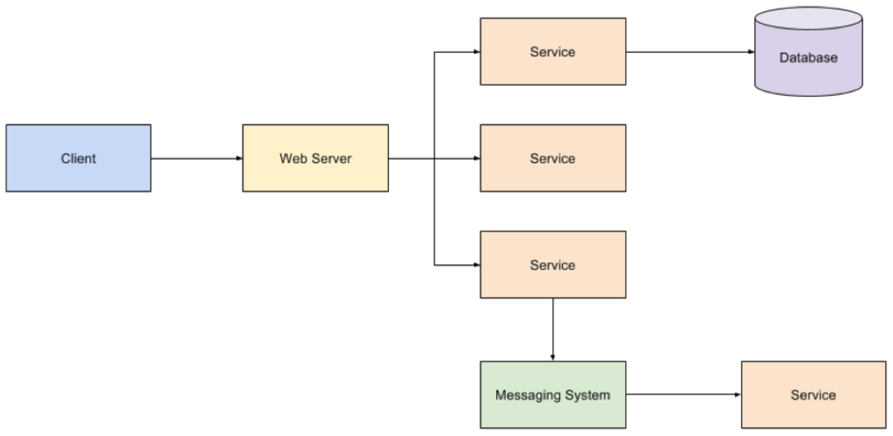


Figure 6. Distributed Application

Figure 6 shows a client making a request to a web server that in turn calls several services. One service makes a database request while another sends a message (via messaging system) to another service. Real systems are complex: single request can touch multiple components running on multiple servers using multiple communication protocols.

TRACING TECHNOLOGIES

With this in mind, how does an APM tool trace a request both within and across services, servers, and data centers?

Inside of a process, an APM tool uses technologies like byte-code instrumentation to intercept calls as they pass from method to method and capture the path that each request follows from when it is received until a response is returned. Figure 7 shows this graphically.

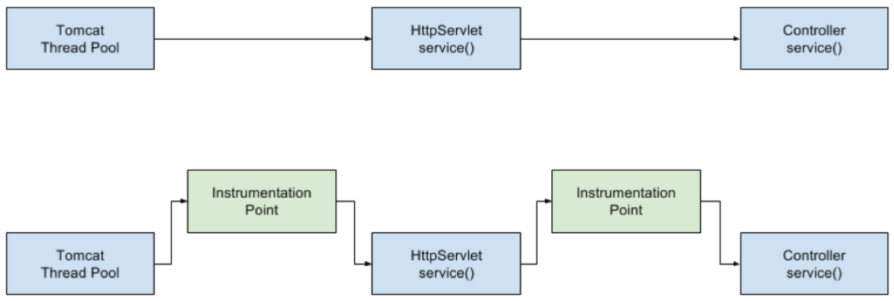


Figure 7. Byte-Code Instrumentation

In figure 7, we present the flow of an application's request handling, alongside a demonstration of how APM instrumentation might inject trace points to get visibility into its performance. You can see that the normal path travels from a Tomcat thread pool to an HttpServlet's service() method and then to a Controller's service() method. When we instrument the request, we intercept the call that Tomcat makes to the HttpServlet service() method and the call that the HttpServlet service() method makes to the Controller service() method. As a result, we can record how long each method takes to execute as well as capture any exceptions that are thrown.

Different tools provide different instrumentation strategies. Some tools instrument every single call and attempt to capture and preserve every trace that occurs. Other tools strategically instrument methods that constitute entry and exit points from the process, typically where something significant is occurring such as receiving a new service call, making an external call to a database, or making an HTTP call to another server. These tools then leverage statistical polling to fill in all of the gaps between these strategic instrumentation points. The goal is to reduce the amount of overhead that the tool adds to the call it is tracing.

Irrespective of the strategy, all APM tools are required to capture and visualize traces within a server so that you can understand the methods that are most contributing to the response time of a request.

Overhead

Overhead is a measure of how much additional resources, such as CPU and memory, that an APM tool adds to an application at runtime. If the APM tool adds an excessive amount of overhead then it may negatively affect the performance of the application, so be aware of the overhead of each tool you are evaluating.

TRACING ACROSS TIERS

In addition to capturing method-level traces inside a server, an APM tool needs to be able to stitch together a request as it passes from one tier to another, or from one server to another. Figure 8 illustrates how this is done.

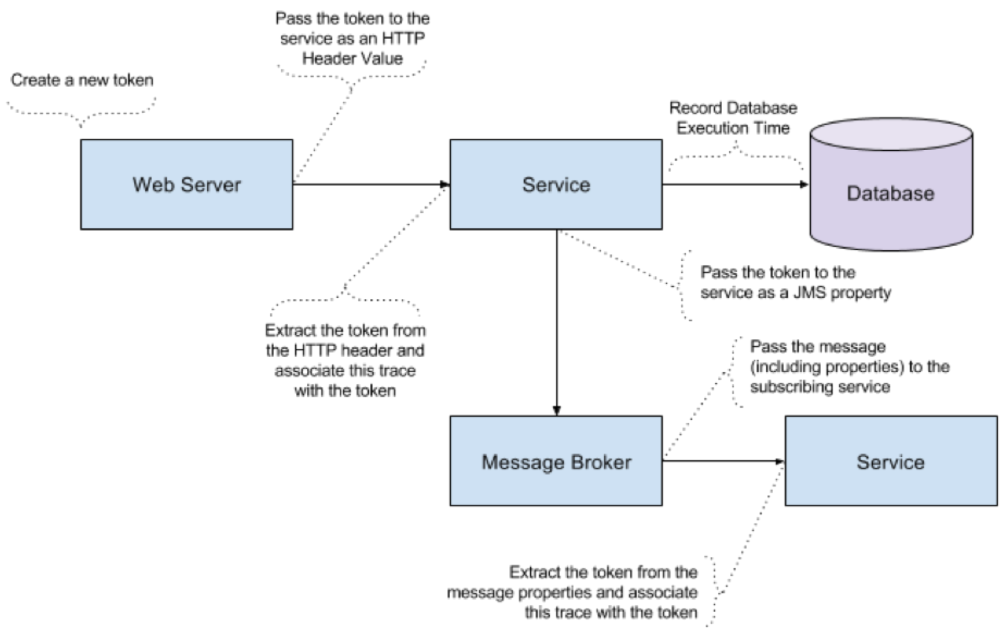

Pass the token to the service as an HTTP Header Value
Create a new token
Record Database Execution Time
Web Server
Service
Database
Pass the token to the service as a JMS property
Extract the token from the HTTP header and associate this trace with the token
Pass the message (including properties) to the subscribing service
Message Broker
Service
Extract the token from the message properties and associate this trace with the token

Figure 8. Tracing across tiers

When the first entry point for a request is received, a unique token is generated to identify the trace. All methods invoked in that first server are associated with that token, but when a request is made between servers, that token is then passed along with the request, using things like HTTP headers and message properties. The receiving server checks to see if there is an existing token and, if there is one, it associates its trace with that token. Finally, when "tiers" complete (web server, services, etc.), then each tier's segment (trace for that request) is sent to the APM tool, which correlates the various segments and constructs the holistic trace.

The result of both in-server tracing and distributed tracing is that when a performance problem occurs, you can quickly identify where the request is spending the most time and the path it followed across your technology stack and logical application tiers.

Automatic vs. Manual Instrumentation

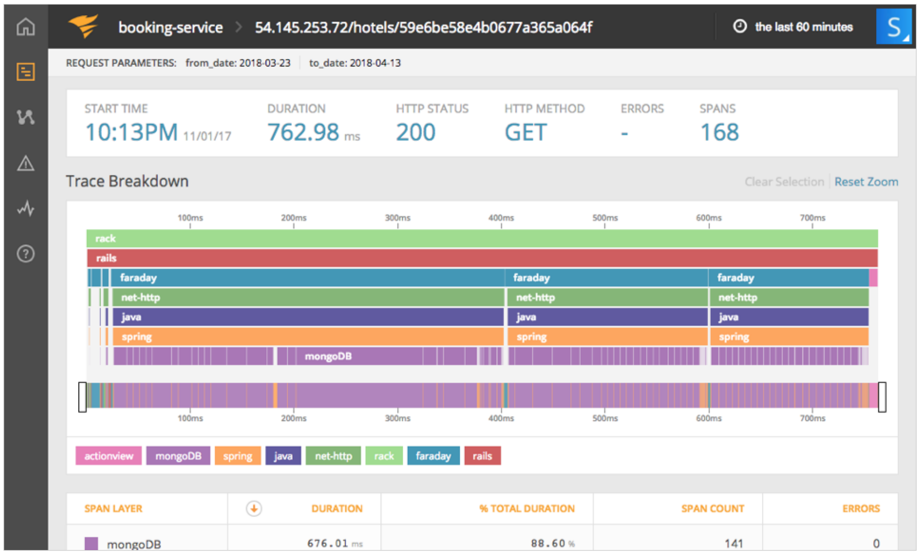
Instrumentation can either be automatic or manual. In automatic instrumentation, the APM tool automatically adds the instrumentation code to your application at runtime, as described above. In "manual instrumentation," you need to add hooks in your application code to identify key behaviors, such as when a request starts and ends or when a database call is invoked. In short, manual instrumentation means that you need to modify your application code to work with your APM tool, whereas automatic instrumentations means your code can stay pristine.

TRACING AND YOUR TECHNOLOGY STACK

In today's modern application architectures and with the adoption of things like microservices, it is important that your APM tool supports all of the technology stacks that you currently use and that you foresee using in the future. Some technologies are better suited to solving different problems and an application may grow organically over time to embrace different technologies. It is therefore important that an APM tool can monitor the complete application and measure how it is performing.

Stated another way, as a request travels from a node service to Java and C# backends and even out to a Golang application, you do not want to switch between monitoring tools. The APM tool should seamlessly show the transition, in the same visualization, from technology to technology so that you can solve your core business problem: identifying the root cause of a performance issue.

TRACING IN ACTION



54.145.253.72/hotels/59e6be58e4b0677a365a064f

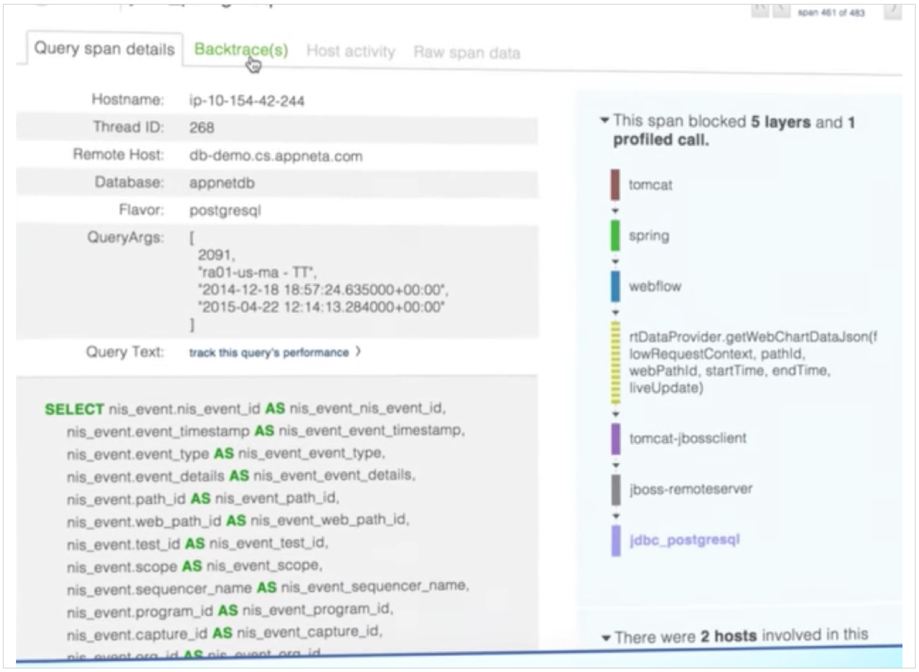
REQUEST PARAMETERS: from_date: 2018-03-23 | to_date: 2018-04-13

START TIME	DURATION	HTTP STATUS	HTTP METHOD	ERRORS	SPANS
10:13PM 11/01/17	762.98 ms	200	GET	-	168

SPAN LAYER	DURATION	% TOTAL DURATION	SPAN COUNT	ERRORS
mongoDB	676.01 ms	88.60 %	141	0

Figure 9. Tracing a request

Figure 9 shows a breakdown of a request by technology stack, with the height of each bar representing the total amount of time. In this example, we see three spikes, all down at the database level (purple). If you click on that bar, you can view the details of the SQL call that is causing the performance issue, as shown in figure 10.



Hostname: ip-10-154-42-244
Thread ID: 268
Remote Host: db-demo.cs.appneta.com
Database: appnetdb
Flavor: postgresql
QueryArgs:
```
[
  2091,
  "ra01-us-ma - TT",
  "2014-12-18 18:57:24.635000+00:00",
  "2015-04-22 12:14:13.284000+00:00"
]
```
Query Text: track this query's performance ›

```
SELECT nis_event.nis_event_id AS nis_event_nis_event_id,
  nis_event.event_timestamp AS nis_event_event_timestamp,
  nis_event.event_type AS nis_event_event_type,
  nis_event.event_details AS nis_event_event_details,
  nis_event.path_id AS nis_event_path_id,
  nis_event.web_path_id AS nis_event_web_path_id,
  nis_event.test_id AS nis_event_test_id,
  nis_event.scope AS nis_event_scope,
  nis_event.sequencer_name AS nis_event_sequencer_name,
  nis_event.program_id AS nis_event_program_id,
  nis_event.capture_id AS nis_event_capture_id,
  nis_event.org_id AS nis_event_org_id
```

▾ This span blocked **5 layers** and **1 profiled call.**

tomcat
spring
webflow
rtDataProvider.getWebChartDataJson(flowRequestContext, pathId, webPathId, startTime, endTime, liveUpdate)
tomcat-jbossclient
jboss-remoteserver
jdbc_postgresql

▾ There were **2 hosts** involved in this

Figure 10. Analyzing a SQL Request

Where did this query come from? You can see the various layers that were traversed on the right, from Tomcat through JBoss®, and down to Postgres, as well as the SQL statement that was executed on the left, including bind parameters.

This level of granularity equips you with the information that you need to start resolving your performance issues.

Temporal Event Management

As mentioned in the beginning of this eBook, one constant in IT is change. You are constantly releasing new versions of your code to production, patching systems, and running backups. The significance of these events to your APM tool is that they will potentially change the behavior and performance of your application. In the simple case of patching systems and running nightly backups, you want to tell your APM tool not to raise alerts that may result from the temporary outage or degradation in performance. Instead, it should make note of new code releases and allow you to view when new releases are published to production.

It is a good idea to compare the performance of an application between releases because it can help you determine if your new release improved or hurt performance. A new release may have new features that add business value to your application, but you need to assess whether those new features impacted the performance of your application in such a way that could hurt your service-level agreements (SLAs).

Furthermore, changes may be subtle and introduce performance degradations that manifest over a long period of time. When you are later investigating a performance issue, knowing when releases were published to production can provide you with a lot of diagnostic insight. Additionally, if you generate good release notes, they can give you hints as to where your performance issue may be and why it was introduced. For example, you may have added a new artificial intelligence capability to handle searches that do not contain specific keywords. In this case, the problem may only start occurring when users learn about your new capabilities and structure their searches accordingly.

The best way to keep on top of changes is to use automation to tie them into your monitoring system. Most applications are built in a Continuous Integration (CI) environment and many are built leveraging Continuous Delivery or Continuous Deployment (CD). These systems detect when source code is checked into a source code management system and then build the code, run a set of unit tests, and in the case of CD, they also run a set of integration tests to validate the build and then potentially publish the application to production. Usually it's as simple as adding one more step to deployment: invoke a service on your APM tool to record the code release. This way, you can track releases without having to remember to do it.

But in order for this to be viable, you need your APM tool to expose an endpoint with which your CI server can integrate.

Intelligent Alerting

Alerting might sound like a simple issue: if something goes wrong, then raise an alert. But, unfortunately, alerting can be tricky. APM tools need to raise alerts, but only when things are truly behaving abnormally. The worst thing that you can do is raise false alerts or raise the same alert over and over.

At the end of the day, "intelligent" alerting means informing you about a problem when it is meaningful to your users. You do not want to wait until the response time is so bad that users are abandoning your site. Rather, you want to monitor metrics that are critical to your users and detect when they are not behaving correctly.

Custom Dashboards

While APM tools define common visualizations that most companies will find useful, they cannot provide every view that will be valuable to you and your organization. As such, an APM tool needs to allow for robust custom dashboards. This means that the tool exposes all of the underlying metrics and has a robust set of graphing tools. Some metrics are better expressed as line graphs, others as bar or stacked bar charts, and some as pie charts, or even simply green/red status indicators.

Figure 11 shows an example application-monitoring dashboard.

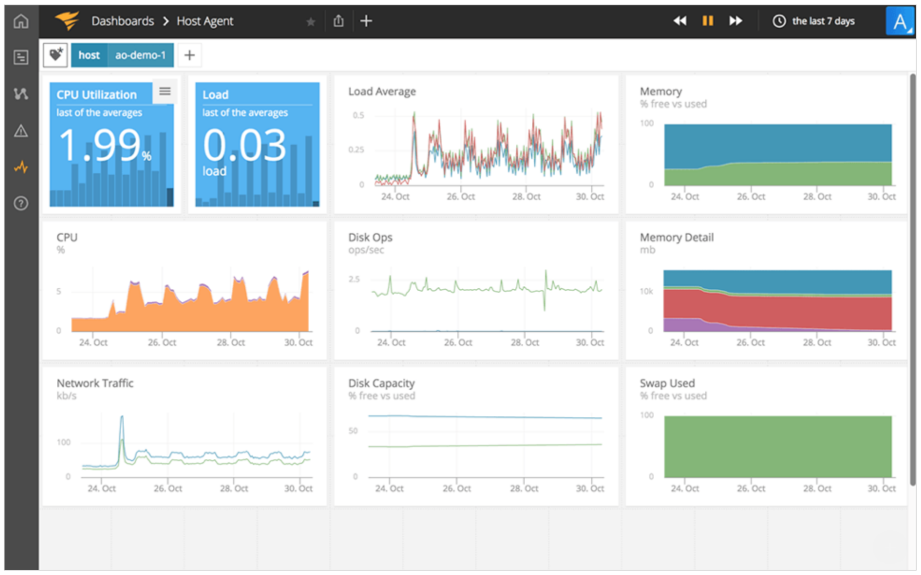


Figure 11. Custom Dashboard

Your APM tool should allow you to easily define views of performance data that are important to your business.

Cost and Benefit

Throughout this eBook, we reviewed the features that make an APM tool an APM tool and explored those features in some level of detail. In this chapter, we are going to focus on APM tools and their return on investment (ROI).

Recall that Gartner breaks APM capabilities into three main segments:

» 1. Digital experience monitoring (DEM)

» 2. Application discovery, tracing and diagnostics (ADTD)

» 3. Application analytics (AA)

And from those segments we derived the following six features:

» Time-series Key Performance Indicator (KPI) metrics

» Infrastructure KPI metrics

» Distributed tracing

» Temporal event management

» Intelligent alerting

» Custom dashboards

Once you have identified the tools with these key features, you have to analyze which ones work best for you and whether or not their differences justify their differences in cost. You also want to choose a solution that meets your current needs and will grow with you. For example, you do not necessarily want to choose a solution that is designed to monitor monolithic applications when you are planning to distribute your application across multiple servers or eventually move to the cloud. Furthermore, you should review APM vendor pricing models to ensure that they are not "stuck in the past." As your application becomes more distributed, the price for your monitoring solution should remain affordable.

APM is essential for your production environment, especially when outages can be measured in dollars lost for every minute that your application is down or for every lost customer because your performance is inadequate.

Some APM vendors charge an exorbitant amount for the same core APM features as everyone else. They'll often disguise the price (and value) with bells and whistles tangential to the main use case of APM. APM shouldn't throw your budget out of whack. It should help you resolve application issues and better serve your end-users. We encourage you to check out AppOptics™, the affordable yet powerful APM and infrastructure monitoring solution.

Check out a 14-day free trial today!