PaaS Cloud for Integrated Full-Stack Monitoring

Application Performance Monitoring systems must navigate challenges and trade-offs to be effective.

By Chandra Krintz and Rich Wolski, Co-Directors, Lab for Research on Adaptive Computing Environments, Computer Science Department, University of California, Santa Barbara

September 2016

Platform-as-a-Service (PaaS) is an increasingly popular technology for deploying Web-accessible applications in the cloud. PaaS serves to hide execution details such as operating systems, resource allocation, network configuration, and service ecosystem management. It also automates load balancing, resource scaling, and fault tolerance. The result for application developers is the ability to focus on programming and innovation and ignore deployment and system issues.

The proliferation of PaaS technology has also intensified the need for new monitoring techniques, as effective monitoring is essential for facilitating auto-scaling, maintaining high availability, managing multi-tenancy (multiple applications sharing resources), and identifying performance bugs in both applications and the PaaS system itself.

Extracting sufficient insight from operating PaaS environments requires that the performance data be extensive and comprehensive, and that the process of data collection be efficient — with little impact on system performance. Therefore, the key to any successful Application Performance Monitoring (APM) system is to effectively resolve these challenges and their associated trade-offs.

Here we propose a new design for PaaS APMs. Instead of an external system (as are most familiar to us), our cloud APM system is tightly integrated with the PaaS cloud. We leverage and augment existing components to provide comprehensive, full- stack monitoring, analytics, and visualization.

This design takes full advantage of the scaling, fault tolerance, security, and control features of existing PaaS platforms while providing a low-overhead End-to-End (E2E) methodology for the monitoring and analysis of cloud applications.

Cloud APM Design and PaaS Integration

Like most system monitoring solutions, our cloud APM features data collection, storage, processing (including data analytics), and visualization.

Sensors and agents, which instrument the applications and core components of the PaaS cloud, collect data. While sensors are physically static in relation to a given component, software agents are more complex and are built to intelligently adapt to changing conditions, including making dynamic decisions about the information captured —and when and how it is captured. Since sensors and agents can impact behavior and performance, they must be lightweight and non-intrusive. To achieve these criteria and assure accuracy, we combine periodic, non-exhaustive sampling with the intelligent placement of sensors and agents throughout the software stack.

For storage and processing of performance data, we leverage the scalable, durable, and highly available distributed services of the PaaS itself. Specifically, our system uses scalable key-value stores, caching systems, relational database systems, high performance search systems, and batch and streaming analytics frameworks to construct the PaaS service ecosystem. For portability across PaaS systems, each function defines an Application Programming Interface (API) through which it interacts with PaaS components and services. For porting to a new PaaS, API operations are rewritten and linked to those of the PaaS.

Figure 1 illustrates APM integration with a typical PaaS stack. The left (dark blue) boxes depict the PaaS architecture. Arrows indicate the flow of data and control in response to application requests. At the lowest level of a PaaS cloud, an infrastructure layer consists of the necessary compute, storage, and networking resources that the PaaS acquires and releases dynamically.

The PaaS kernel is a collection of managed, scalable services, which implement the common functionality required by most cloud applications. Application developers innovate by composing these services, which typically include data storage and caching, queuing services, authentication, user management, and many others.

Increasingly, PaaS clouds provide a managed set of APIs (Cloud SDKs) used by developers to link PaaS functionality to applications. Cloud SDKs — like similar PaaS proxy mechanisms — simplify, control, and load balance access to PaaS services across applications and the system.

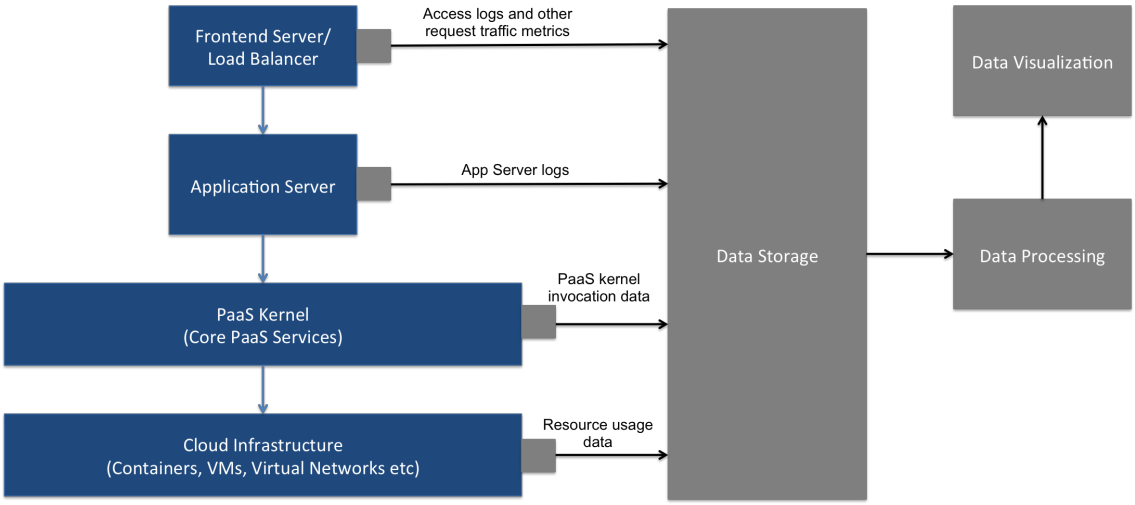

Figure 1: APM Architecture

Application servers execute copies of an application and link the application code with the underlying PaaS kernel. The servers also isolate and sandbox the application code for secure, multi-tenant operation and to provide control and charging for services. Load balancers (or front ends) serve as the entry point to applications, intercepting application requests and filtering and routing them to the appropriate server instances.

Back to Figure 1, the small grey boxes attached to the PaaS components represent the sensors and agents employed to monitor the cloud platform components to collect events and performance data. The APM collects data from all layers in the PaaS stack (full-stack monitoring). The rate at which measurements are taken is configurable and, in many cases, adaptive; moreover, sensors and agents use batch operations and asynchronous communication to reduce performance disruption and overhead.

From the frontend and load balancing layer, the APM gathers information related to incoming application requests. Monitoring agents scrape HTTP server logs to extract timestamps, source/destination addresses, response time, and other parameters. This information is usually readily available for harvesting as frontend technologies, such as Apache HTTPD and Nginx. Additionally, agents collect information about active connections, invalid access attempts, and HTTP errors.

Within the application server layer, sensors collect application and runtime/container log data, which typically includes process-level metrics indicating the resource usage of individual application instances. Targeting log files avoids the high overhead of application and application server instrumentation, which can be added via additional agents if the benefits warrant the extra overhead.

Within the PaaS kernel, we instrument the entry points of *all* PaaS services:

- Collecting caller and callee information
- Timestamp
- Measured execution time per operation invocation
- Request details, including size and hash of the arguments

This helps distinguish different phases of PaaS execution and aggregate and characterize operation invocation instances. It also enables low overhead yet accurate full-stack monitoring.

From the cloud infrastructure, information related to virtual machines, operating system containers, and processes and resource usage of each physical host machine is collected. Most sensors at this level scrape logs and query the Linux profile system, gathering metrics about the network, CPU, memory, and

resource allocation and reclamation decisions made by the management and orchestration frameworks.

The collected information enables clustering-related activities and events across the system. Since PaaS systems commonly host Web applications and services, our APM design considers web requests as events. A request identifier is attached to each HTTP request header, which is visible to all components. The appropriate APM agents are then configured to record these identifiers. The data processing layer then clusters measurements by request identifiers to facilitate E2E system analysis for individual Web requests.

The data processing layer stores and provides scalable access to this performance data. It also permits plug-in analysis routines that can be used to characterize application and system behaviors over time; detect behavioral and performance anomalies and workload changes; and identify opportunities for more effective resource utilization and automatic scaling of resources, services, and application instances.

These analysis routines perform both inference and prediction and make use of statistical analysis libraries, batch processing services, and search and query systems.

ElasticStack as the APM foundation

After a thorough evaluation, we chose ElasticStack to be the foundation for APM. ElasticStack is an open source distributed system built on Linux KVM (for Kernel-based Virtual Machine) for data storage and processing.

ElasticStack comprises three main components: ElasticSearch, Logstash, and Kibana:

- ElasticSearch supports scalable and highly available management of structured and semi-structured data via automatic sharding and replication. It also provides comprehensive data indexing, filtering, aggregation, and query support to simplify the implementation of our high-level data processing algorithms. ElasticSearch is easily integrated within popular Big Data workflows such as Spark and MapReduce for advanced analysis.
- Logstash facilitates data extraction from a wide range of standard log formats. Custom log formats are supported via simple configuration.
- Kibana provides a powerful Web-based dashboard and a wide range of charting, tabulation, and time series support.

In addition, custom data processing and analytics components, as well as extensions, can tailor the visualization of data to the metrics under interrogation, which makes it easier to extract actionable insights.

APM Use Cases

The following use cases employ the PaaS performance data collected by our APM to provide new PaaS features. Of particular interest is how our APM system can help predict performance-based Service Level Objectives (SLOs) for Web applications deployed in a PaaS cloud, and how performance anomalies are detected across the PaaS stack.

Application Response Time Prediction

This APM use case provides scalable and accurate response time predictions that can be used between a cloud provider and PaaS user as a per-application SLO. To enable this, we combine static program analysis of the hosted Web applications and APM monitoring of the PaaS cloud. Because we want to provide the prediction to PaaS users when they are deploying the applications, we perform this static analysis immediately prior to deploying or running an application on the PaaS cloud.

For each path through a function, our static analysis extracts the list of PaaS kernel calls (invocations and access to PaaS services) via traditional techniques for abstract-interpretation-based loop bounds analysis, branch prediction, and worst-case execution time analysis. To save overhead, we do not instrument the application to collect performance metrics at runtime. Instead, the lists are recorded in the APM system and services are monitored in the system independently of the application's execution.

In particular, the system employs the PaaS kernel services performance data collected by the APM. An analysis routine then extracts a time series of operation execution times for each service in the list. A forecasting methodology then calculates statistical bounds on the response times of applications. These

values are then used by the cloud provider as the basis for a performance SLO.

To make SLO predictions, we use Queue Bounds Estimation from Time Series (QBETS), a non-parametric time series analysis method we designed for predicting the scheduling delays of batch queue systems in high performance computing environments. We adapt it for use 'as-a-service' in our PaaS APM system to estimate the response time of deployed applications.

Because PaaS service and platform behavior under load changes over time, our predicted SLOs may become invalid in time. Our system detects SLO violations so that they may be renegotiated by the cloud provider. When such invalidations occur, the PaaS invokes our SLO analysis routine in the APM to establish new SLOs.

We have integrated our Cloud APM within the Google App Engine and have also integrated it into the full stack of the private open source PaaS AppScale. We use these platforms to perform extensive testing and empirical evaluation of open source Java Web applications that execute over them. We find that our system generates correct SLOs in all cases.

Performance Anomaly Detection

Numerous statistical models have been developed for detecting performance anomalies dynamically; however, most prior work focused on simple, stand-alone applications. Our goal is to provide anomaly detection for PaaS-based (distributed) Web applications. For this, we implement multiple APM analysis plug-ins called *anomaly detectors*. These processes periodically analyze the performance data for each deployed application in the PaaS.

Multiple detector implementations each use a different statistical method for detection. Detectors at the application level allow different applications to use one or more different anomaly detectors, each with an execution schedule and a sliding window of singly processed data; for example, from 10 minutes ago until now.

In addition, a *path anomaly detector* leverages the PaaS kernel call list for each request processing path through an application; however, in this case the data gathered from the PaaS kernel instrumentation (PaaS kernel invocation data) is used to infer the execution paths for individual applications. The detector computes the frequency distribution of different paths and detects changes in this distribution over time, identifies the occurrence of new paths, most frequently executed paths, and significant changes in path frequency distribution.

Upon detection of an anomaly, the detector sends an event to a collection of anomaly handlers. The event encapsulates a unique anomaly identifier, timestamp, application identifier, and the source detector's sliding window corresponding to the anomaly. Anomaly handlers are configured globally but can be configured to ignore certain types of events. As with detectors, the APM supports multiple anomaly handlers: one for logging anomalies, one for sending alert emails, one for updating the dashboard, and so on.

Additionally, we provide two special anomaly handler implementations: a workload change analyzer and a root cause analyzer:

- The workload change analyzer analyzes historical workload trends via a suite of change point detection algorithms.
- The root cause analyzer evaluates the PaaS kernel call historical trends and attempts to determine the most likely components of the cloud (in the PaaS kernel) that may be attributed to a detected anomaly.

Anomaly detectors and anomaly handlers work with fix-sized sliding windows, discarding old data as the sliding window moves along the timeline. Since the amount of state information these entities must keep in memory has a strict upper bound, the processes are lightweight. Historical data can be persisted in the APM for offline batch processing, if needed.

Conclusion

As PaaS use grows in popularity, the need for technologies to monitor and analyze the performance and behavior of deployed applications has become essential; however, most PaaS clouds do not provide adequate support for lightweight, full-stack performance, data collection, and analysis.

Many monitoring frameworks have been designed to support gathering and analyzing performance data to draw insights about system behavior, performance, availability, and faults. Unfortunately, while many support data collection, storage, analysis, and visualization to varying degrees, none are designed to operate as part of a cloud platform. Data storage mechanisms, APIs, and configuration models are targeted at monitoring servers or applications as individual entities and do not support E2E tracing of request flows in a larger system. Further, they are not easily extensible, support only basic metric calculations, and provide no support for correlation or root cause analysis.

As a solution, we have proposed a new, easily integrated APM system that can take advantage of PaaS cloud features for comprehensive, full-stack monitoring and analytics. Our APM can be integrated into a PaaS system by customizing a set of API calls and can facilitate inference and prediction. Such functionality can be used to guide new PaaS services, including response time SLOs at application deployment time, system wide performance anomaly and workload change point detection, and root cause analysis for application performance anomalies.