Office of the CTO Perspective: Cloud Distributed Application Runtime–

An Emerging Layer for Multi-Cloud Application Services Fabric

vmware

OFFICE OF THE CTO

Table of contents

1. Introduction

On May 2019 we released a paper that discussed the various challenges of multi-cloud application platforms, and how the concept of a cloud runtime can address those challenges. A cloud runtime is a concept of common application services connectivity with various resiliency and geographic manageability functions. The concept begins with a premise: regardless of where your application runs, the cloud runtime must provide a set of OSI Layer 7 application platform resiliency and manageability services that improve the overall quality of service—and, ultimately—the user experience.

As an application owner, you should be able to trace the overall performance of user transactions in specific geographic locations, from the time a user clicks on an application to the time the transaction is completed, regardless of the number of distributed cloud services that transaction uses. More importantly, instead of contending with multiple "tuning knobs" to achieve such quality of service, the cloud runtime gives you the ability at deployment time to state the service level objectives (SLOs) and service level indicators (SLIs) you expect from the system. The cloud runtime is then bound to adhere to your preferences.

The basic principles of cloud runtime are: end-to-end traceability of application transactions from edge to cloud; geographic distribution of application services based on capacity and proximity rules; a consistent observability perspective; various resiliency functions, such as autoscaling, circuit breaking, and cloud bursting; and a declarative model that is SLO driven.

We refer to "cloud runtime" as a runtime because it acts as a common cloud abstraction layer for application services. Such services receive various platform-level functionality that improves ongoing deployments and resiliency of distributed applications overall. From an abstraction perspective, we build on three main layers–Kubernetes, service mesh, and cloud runtime. Kubernetes forms the bottommost layer, service mesh is the next layer, and then cloud runtime sits on top of service mesh to offer the various quality of services functions.

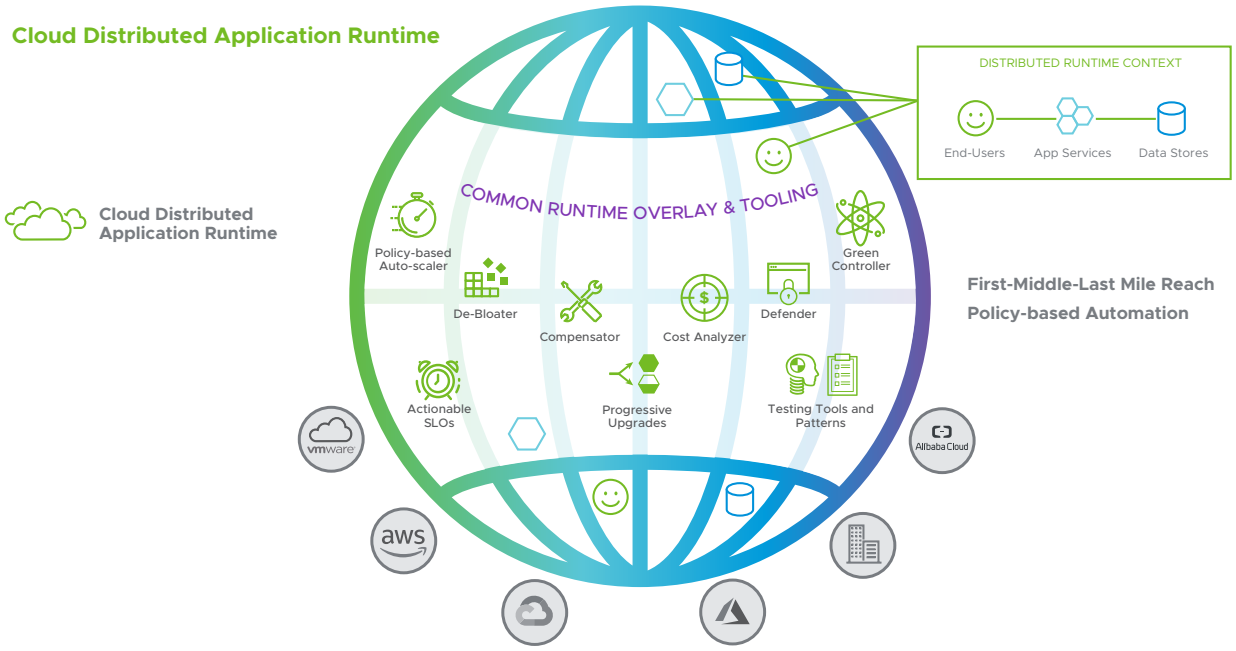

FIGURE 1: Cloud Distributed Application Runtime.

Figure 1 shows a high-level diagram of the cloud runtime concept. The diagram highlights the various functions offered by the runtime, as outlined below:

- **Cloud Runtime Context:** This allows for the full round-trip traceability of user actions in a multi-cloud environment. Cloud runtime context will contain a distributed span id carried forward to all participating services, per user request.
- **Policy-Based Autoscaler:** This gives the user an ability to scribe an autoscaling configuration with metrics and thresholds that will trigger scaling or descaling actions. In its most generic form, the system can track the behavior of a user-scribed Sloan scale or descale as needed. The Tanzu Service Mesh (TSM) Autoscaler currently under development, which we sometimes refer to internally as a

Predictable Response Time Controller, takes as its main motivation to give the user an ability to control the distributed system with some predictability of application response times. For further details on the various features of the developed algorithm for the TSM Autoscaler, refer to TSM Autoscaler.

- **Declarative Intents with SLOs:** This is a formalized way to describe, measure, and monitor the performance, quality, and reliability of microservice applications in a multi-cloud deployment of distributed applications. SLOs provide a shared quality benchmark for application and platform teams to reference for gauging service level agreement compliance and continuous improvement. An SLO describes the high-level objective for the acceptable operation and health of a service or application over a length of time, such as a week or a month. Operators can specify, for example, that a service or application should be healthy 99.99 percent of the time. The runtime will support monitored SLOs that basically track the overall SLO, but also offer actionable SLOs that will trigger certain resiliency functions based on the set SLIs. It can trigger autoscaling, descaling, or cloud bursting, check available capacity before triggering any action, and fire. It can even trigger a warning event, too—even circuit breaking, if it deems the situation appropriate.
- **De-Bloater:** This controller looks for opportunities to reclaim unused, or just lazily used, container services. Since the cloud runtime has all the metrics on service performance, it can make various recommendations about improving the scale factor. For example, the controller can count the number of instances, the request volume, and the total compute space utilization and make recommendations for reducing the number of instances to fewer vertically larger instances. These calculations would be based on historical data and various utilization ratios.
- **Compensator:** This is a controller that allows for the registration of an error type and the accompanying error remediation handler—a handler in the form of either a microservice or an internal/external function. Over a period of time, a service operator can register many such remediation handlers to build a dictionary set of their mindshare about the ongoing upkeep of the service. Instead of such remediation handlers being triggered manually in external systems, the cloud runtime can trigger these via the compensator pattern, and offer opportunities to track execution, frequency of error, success of remediation, and various other metrics associated with the action.
- **Progressive Upgrade:** This controller deals with initial deployment and ongoing updates of services within the runtime. The controller will pick the cluster with the correct proximity and compute space availability of the application cluster or namespace. For ongoing upgrades, the controller offers progressive rollout strategies such as blue-green, canary, A/B testing, or shadow mode, and would assess the impact of such deployment on the committed SLO for the service. If the current SLO error budget is too depleted to allow for an update, the controller will raise a warning.
- **Cost Analyzer:** Through shadow-mode simulations, an application deployer should be able to simulate the cost of running their application services on the cloud runtime and get an approximate cost of completing transactions.
- **Defender:** The main premise of this controller is to provide integration with northbound Web Application Firewall (WAF) function. For example, if the WAF detects there was an SQL injection attack, it can inform the Defender and block the attack. In turn, the Defender can inform all interested parties via various notification mechanisms to trigger further remediation actions. The remediation actions can be proactive about double-checking the state of an application to ensure that the attack was stopped in time or run various analytics to determine the extent of the attack. Beyond SQL injection, the Defender can integrate and take advantage of various WAF signals to better improve security and resiliency of applications.
- **Testing Tools and Patterns for Chaos Controller:** This allows the user to roll out new services in a mesh and describe a randomized fault injector. For example, the user may say "randomly fail 10 percent of my services." The controller will action this and measure the overall SLO of these services and compare them to a baseline. The user can then compare multiple chaos tests and use this data to drive an improvement to the resiliency architecture of the applications.
- **Green Controller:** Leverages SLO policies to drive sustainable resilience patterns–for example, the policy-based autoscaler can be placed in efficiency mode, where it can scale up or down while adhering to the overall SLO.
- **NOTE:** We also referred to the cloud runtime concept as Hybrid Cloud Runtime in a recent patent filing.1 Here is an excerpt from this patent: "In some embodiments, the performance controller 170 is a component of a Hybrid Cloud Runtime configured to manage one or more hybrid cloud-computing environments (e.g., the private cloud environment 102 and the public cloud-computing environment 104) in the hybrid cloud system 100."

2. Historical Background and Motivation of Cloud Runtime

2.1 From service mesh to cloud runtimes as a new emerging pattern

We often meet with CIOs who are struggling to maintain desired SLOs for their users. Their systems suffer from lackluster reliability and have a complicated performance posture. When they ask their teams for an explanation of why systems are not meeting SLOs, it turns into a complicated and convoluted discussion of, "Yes, we have more knobs for that problem too." But what CIOs really want are fewer knobs and more intelligent systems that can observe application performance and replace the multitude of knobs with automation. Many want their systems to behave in accordance with the prescribed SLO, where the system interprets and adheres to an SLO in an automated way.

Today, we are far from this ideal world in the enterprise. Many SLOs are defined in spreadsheets, lying dormant, with various manual processes to stitch them together in order to (hopefully) achieve the desired systems behavior. In many cases, organizations forget what the defined SLOs are because they are contained in a spreadsheet that is not actively maintained or referenced. This flawed approach creates an organizational disconnect and fails to encourage collaboration among teams.

In this section, we examine the "CIO Conundrum," look at how we can make multi-cloud platforms more reliable, leverage SLO-based mechanisms to tame multi-cloud services, look at what new abstractions are emerging, and offer a market perspective of what the cloud-native community is dabbling in. Note: This is not the first time our team has written about SLOs. You may want to read SLOS — The Emerging Universal Language in the Enterprise and Application Platforms Position Paper.

2.2 The "CIO Conundrum"

In Figure 2, we depict the vicious cycle in which CIOs are caught. A missed SLO results in a response to over-provision hardware—which is necessary, because there is a gap in knowledge regarding how to properly implement an application-level scalability solution. Upon closer inspection of this knowledge gap, we often find that it is due to various organization silos that exist in the enterprise, as well as the fact that no particular silo owns the end-to-end SLO. The typical debate is: are the problems in the application code, application runtime, or platform/compute space? The solution could be in any of these three parts, but often the various organizations involved have varying perspectives on the solution. These varied perspectives drive the growth of organizational silos and extend the gaps between them even further—and this, of course, causes the inability to reach a proper solution.

If CIOs were only over-provisioning cloud infrastructure, they would only have to explain to their CFO why such stable systems cost so much to run. However, these systems cost more to run and are also unreliable, complicating the conversation. Missed SLOs have a direct impact on the business, and they lead to poor customer experiences that complicate a CIO's life. There must be a better way!

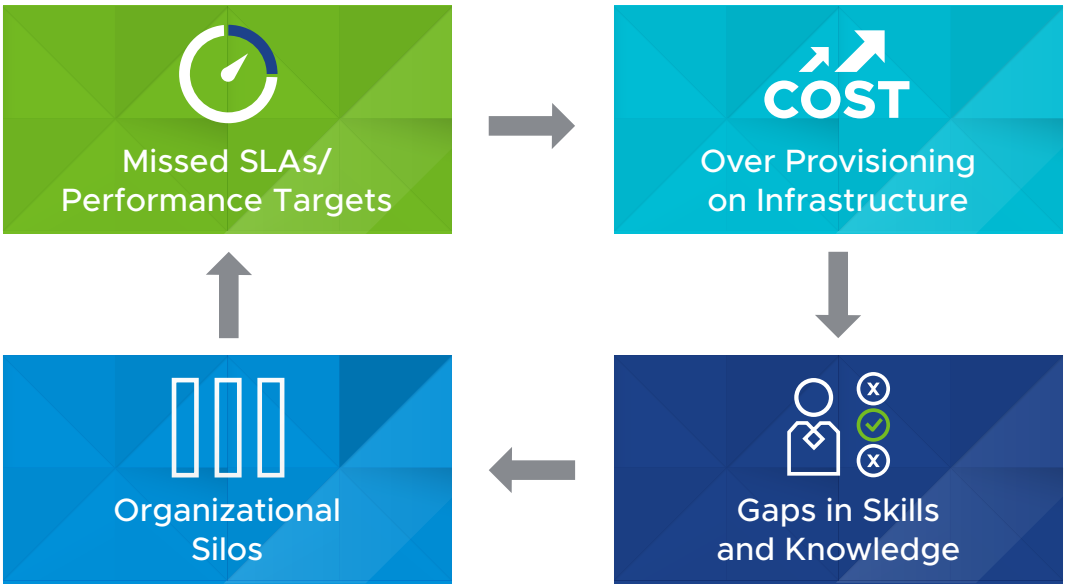

Figure 2: CIOs stuck in a vicious cycle

2.3 What CIOs really need is continuous sharing of common information across the silos

In a recent conversation with customers, they acknowledged that silos complicate their system designs, implementations, and operations, and even further divide the knowledge gap between their teams. What is even more interesting, however, is what the customers' CIO highlighted: When there is an outage, all teams come together to share information in real time, using data to drive immediate decisions. But once the outage issue is resolved, all the teams fall back to their silos.

This behavior indicates that organizational silos will continue to exist, as they provide a useful structure for getting work done. What is really critical, though, is the ability to have shared information across the silos. Systems can act as an overlay to provide just-in-time data for making decisions all the time, regardless of the organizational silos. That is the key. It is not that you really want to break down silos. After all, if you break down silos today and wait a few months, new silos will appear. What is important is having a free flow of common information between the organizational silos to get them to collaborate in real time. It is this shared, real-time information that can help drive better reliability across cloud platforms and services in the enterprise.

But what does it take to make the silos more efficient? This can mean many things: first, it means more continuous information flow, less debate over which metric makes sense, and a drive towards better root-cause analysis. CIOs are looking for their teams to learn from these root-cause reports and build more capable systems. CIOs need a way to automate their systems to interpret SLOs, measure and analyze the metrics that affect SLOs, and automatically act to achieve the SLOs. Ideally, all this would occur via a unified system.

CIOs expect cloud-application platforms to deliver SLOs in a reliable fashion, in much the same way robots in the auto industry use real-time information and encoded steps to improve the throughput of automobile manufacturing (see Figure 3). SLOs will provide an understanding of what one system can reliably expect from another and then based on that understanding, create a cascade system of reliability trust to build more complex features that will accelerate the business.

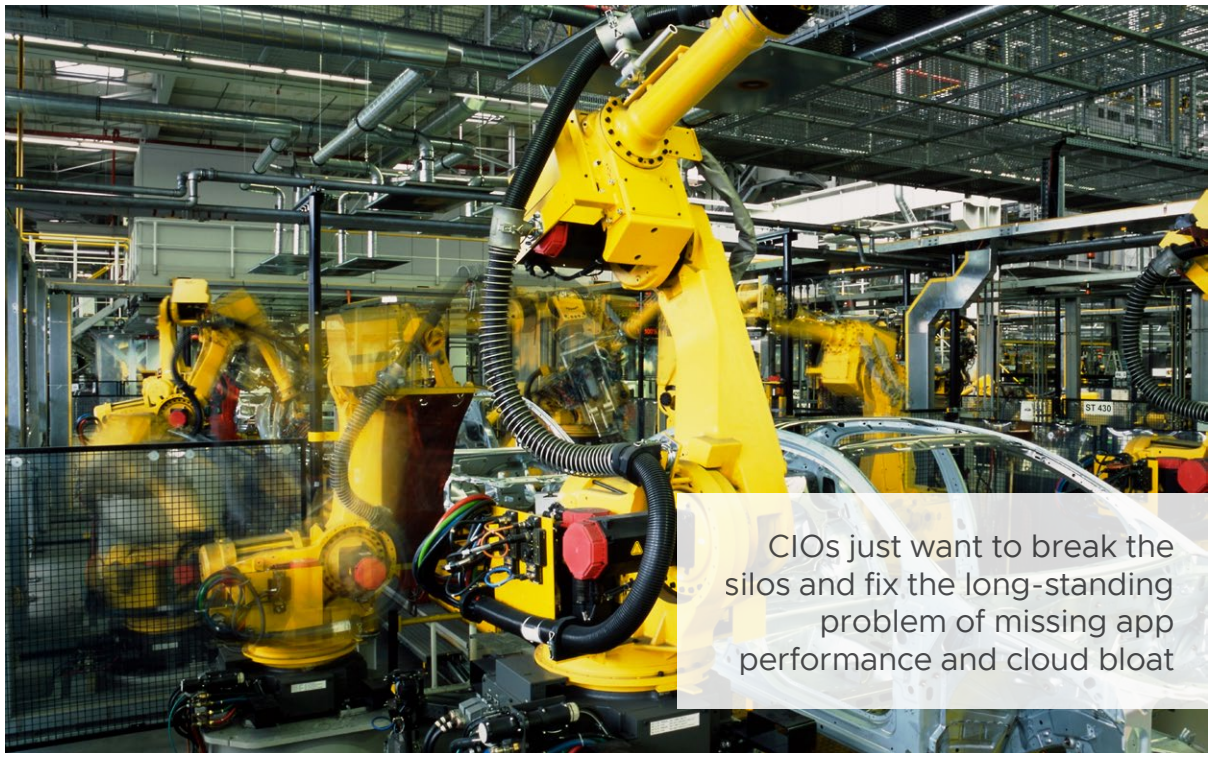

FIGURE 3: Robots rely on real-time information—as well as the SLOs of other robots—to complete a chain of automated work. The result: a distributed system of trust, gated by a cascade of SLOs that everyone relies on. Distributed cloud services with SLOs must behave in a similar manner.

2.4 So, are we alone in our way of thinking about SLOs and how they can drive improved reliability? Recent KubeCon sessions

In our recent visit to KubeCon 2019, as shown in Figure 4, we saw that approximately 33 percent of the sessions had something to do with SLO/policy/intent-based approaches to controlling distributed services. Most of these sessions discussed how large multi-cloud distributed services involve huge scale-out factors and spawn the realization that you need effective real-time telemetry with dynamic control actions to achieve any stated SLO/policy/intent. No doubt these systems are like graphs, and it is becoming harder to make sense out of them.

The best we can do is measure metrics in real time, interpret these metrics, and then build systems that can understand the metrics and take appropriate control actions. Essentially, we are building robotic systems where all systems share the same real-time information—and, in a coordinated fashion, act in concert to meet their agreed upon objectives.

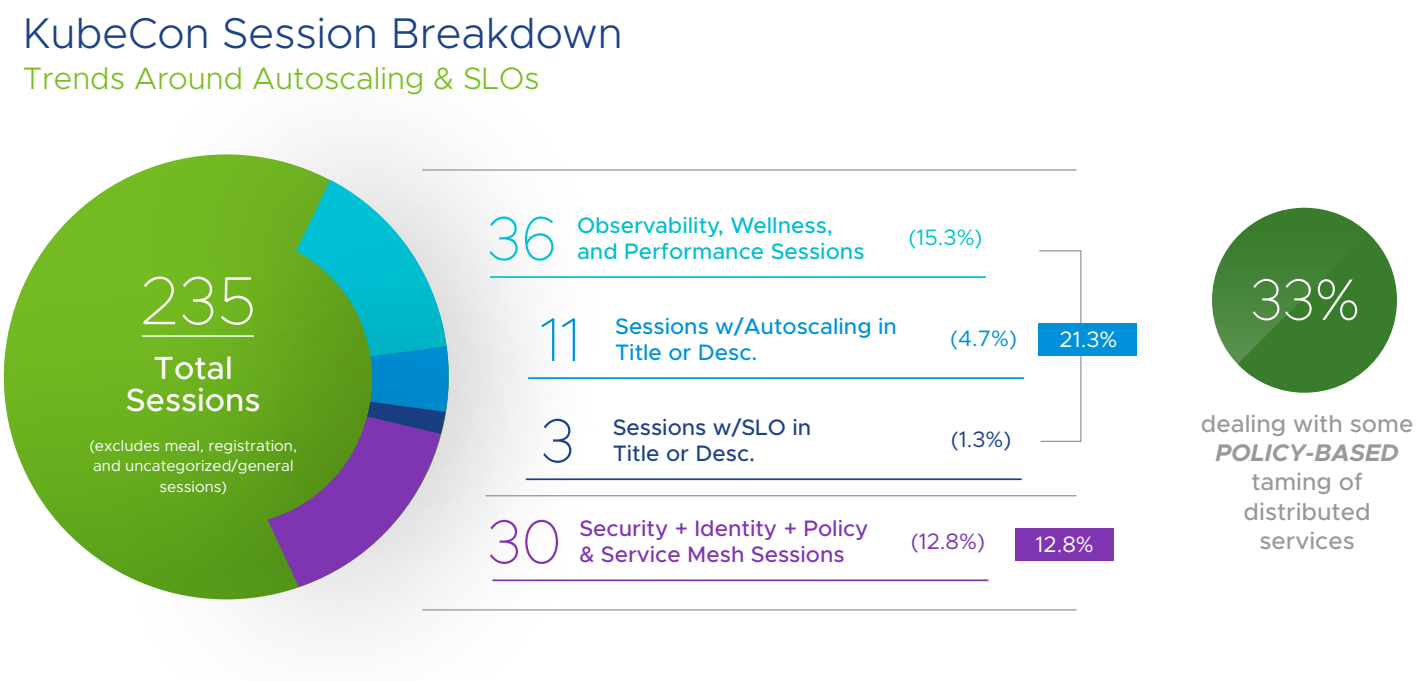

FIGURE 4: More than a third of the recent KubeCon sessions were focused on SLO/policy/intent auto-scaling and reliability mechanisms.

2.5 The emergence of cloud runtimes: new abstractions driving next-level application resiliency

Thus far, we outlined the "CIO Conundrum" and how SLOs can help alleviate the vicious cycle brought about by organizational and technology-design silos. In this section, we look at the challenges that CIOs must address related to technology teams, in order to deliver end-to-end solutions that guarantee quality of service (QoS) for applications and compelling experiences for customers. We will also discuss how such systems are being built today—to guarantee QoS from the time a user initiates a transaction in an application until the transaction is fully executed across a set of multi-cloud distributed services.

2.5.1 How the end-to-end user experience is handled in the enterprise today

In order to deliver on end-to-end QoS and SLOs today, we cobble together metrics from three different teams—namely the Application Platforms Team (APT), Data Platforms Team (DPT), and User Experience Tools Team (UETT) (see Figure 5). These are the three main teams within the enterprise that have some level of responsibility to deliver on the QoS for applications. For simplicity, we have omitted other teams that may play a role.

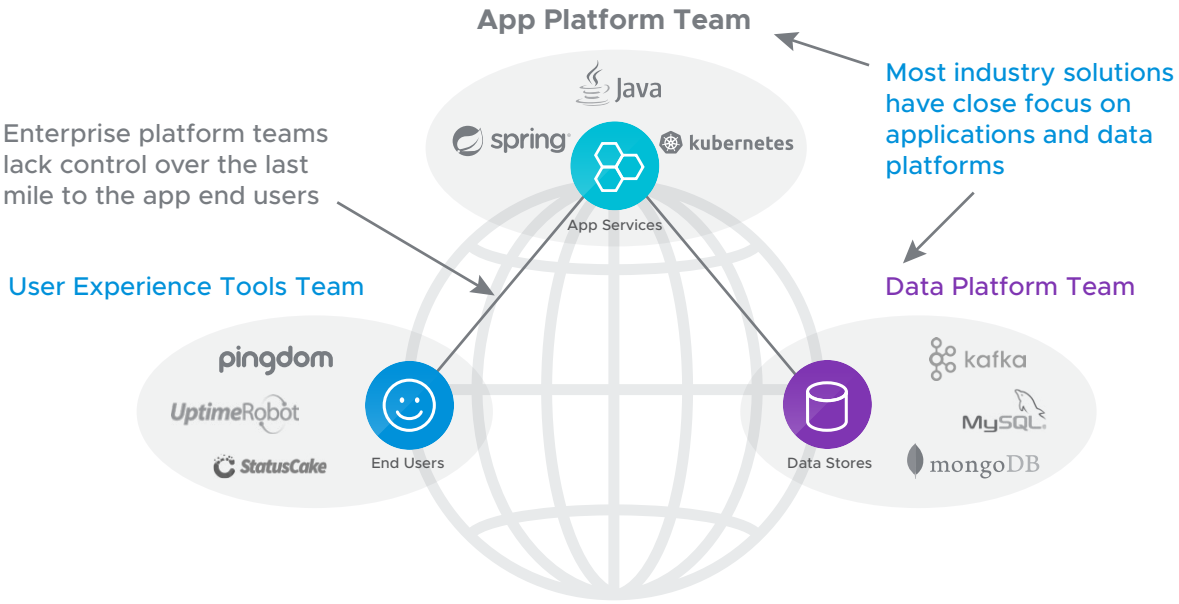

FIGURE 5: This diagram illustrates how three key teams are involved in measuring end-to-end quality of service for multi-cloud applications, including managing availability, resiliency, reliability, and feature velocity at scale.

Let us assume that these three teams are working together on running a mobile banking application that provides various personal banking transactions, such as depositing checks, transferring funds between accounts, paying bills, and so on. To guarantee the end-to-end SLOs of this banking application, the teams have to build a system that can provide an answer to the following question: What response times are users located in New York experiencing at any point in time (or any other location where the bank does business)?

Systems capable of providing such answers must gather telemetry, metrics, and tracing data from the time a user clicks on their mobile application to initiate a transaction, to the time the transaction completes. The tracing would provide visibility into the entire transaction as it propagates across distributed services/APIs running in different clouds. This implies gathering spans and metrics from all three teams mentioned earlier (see Figure 5): the UETT, APT, and DPT. These teams must collaborate to stitch together dozens or hundreds of spans in order to provide end-to-end tracing and tracking information—along with an ability to analyze the user experience and take corrective actions to improve it in near real time. Not an easy undertaking.

For example, the UETT is concerned with using edge-based services to geographically ping the banking application (as shown in Figure 5). An example of this is UptimeRobot, or other similar tools, which simulate clicks and initiate synthetic transactions to gather information on response times, or other metrics deemed important to the SLOs being tracked.

The UETT would then gather and share the response-time telemetry information with the APT so they can conduct further analysis. It is here with the APT where things start to become interesting, as they try to dissect and correlate what the response time information means. This is no easy task, and it requires a lot of plumbing tools (that is, internal tools combined with external vendor tools and services) to enable full transaction tracing. The APT will also need to trace back to the data backing services provided by the DPT. Finally, the data may need to be correlated for additional analytics. Correlations can drive control actions such as scaling, circuit breaking, redirecting, and other resiliency patterns, which improve the end user experience all in near real time.

Having worked on many such systems with our customers, we know there is a tight integration between the APT and DPT teams, and a high degree of use case continuity from various solutions and products available in the market. However, the gap between the APT and UETT teams is where things start to fall apart. Based on our observations, the gap between the APT and UETT is wide, largely due to differing expertise/skill sets, processes, and tools.

The gap between the APT and UETT teams creates a challenge, of course, in terms of implementing elegant solutions at speed and scale. For example, how do you accommodate delays in receiving metrics from various systems, adapt/convert different metric formats, work with multiple scripting languages, and issue real-time control actions to deliver on the QoS and SLOs? Today, these challenges are unsatisfactorily addressed with many tradeoffs and compromises by applying additional manual—and costly—effort.

2.5.2 How have systems been coping and delivering end-to-end SLOs?

Adding to the complexity of handing off telemetry and metrics between teams in near-real time, is the fact you need to do this in a geographically distributed manner. Most organizations do not have cloud infrastructure in every possible geographic edge, so the logical solution for most organizations has been to leverage a cloud and/or a content delivery network service to take the application services and associated SLO measurement tools to the edge. For example, Lyft runs Envoy proxies distributed all around the world, and leverages VMware Tanzu Observability by Wavefront to handle the telemetry stitching, providing near real-time visibility into their ridership metrics, and using these to optimize the application experience.

Gathering metrics and telemetry is a critical part of the solution, but the other aspect is to use these metrics to issue control actions that can help improve the user experience. This is where companies such as Lyft have spent quite a lot of time developing specific custom control layers that analyze telemetry data and take policy-based actions to ensure applications meet desired SLO targets.

2.5.3 What could these types of control systems look like in the future?

So far, we have talked about the complexity of setting up end-to-end metrics, traceability, and SLOs. Is it possible to improve the user experience with a purpose-built SLO-driven system that can trace user transactions from the edge across a distributed application service graph? And then take control actions such as scaling, circuit breaking, redirecting, and other resiliency patterns in near-real time? How can we abstract these control functions into a common reusable layer that is consumable by applications?

One way of abstracting these concerns away from the application business logic and the underlying cloud infrastructure is to leverage a sidecar proxy pattern and service mesh controllers. However, a service mesh alone is not enough. Additionally, we need to have specialized control layers on top of a service mesh to abstract a common set of application resiliency services. This notion of a common layer, referred to as a cloud runtime, was first mentioned in the VMware OCTO Application Platforms Position Paper.

3. References

[1] Attorney Docket No. F718

METHOD AND SYSTEM FOR PERFORMANCE CONTROL IN A CLOUD COMPUTING ENVIRONMENT - Emad Benjamin, Michael Gasch, Daniel Linsley, Frank Carta, Greg Lavender

Benjamin, Emad, and Michael Gasch, Daniel Linsley, Frank Carta, and Greg Lavender. Method and System for Performance Control in a Cloud Computing Environment. US Patent pending.

vmware

OFFICE OF THE CTO

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 vmware.com Item No: OCTO_Cloud_Runtime_R2 4/21