Observability for Modern Application Platforms

Introduction to Cloud Observability Frameworks

Version 1.0 - Wednesday, September 16, 2020

vmware®

Table of Contents

Executive Summary 3

Summary Overview of this Paper 4

- Telemetry Data and Feedback Loops 4
- Cloud Observability Framework 4
- Business Benefits 5
- Current State of Observability 5

Drivers for Observability 6

Relationship Between Observability and Monitoring 6

Addressing Organizational Complexities 8

- Areas of Responsibility and Visibility 8
- Site Reliability Engineering and DevOps 9
- Service Level Objectives and Personas 10

Observability for IT Organizations 10

Cloud Observability Framework 12

- Telemetry Data Pipelines 14

Instrumentation for Observability 14

- Types of Telemetry Data 15
- Metadata 16
- Observability Data Types 16
- Events Versus Logs 17
- Topology and Inventory Data 17
- Instrumentation Deployment Considerations 18

Observability Use Cases 18

- Personas 19
- Key Performance Indicators and Service Level Objectives 21
- Metrics Collection 21
- Log Aggregation 22
- Alerting and Incident Management 23

Current State of Observability 24

Conclusions 25

References 26

Acknowledgments 27

Executive Summary

This white paper presents an overview and reference framework for observability, a new technology focus area that is gaining industry traction as part of the enterprise migration towards the cloud. Observability, together with data analytics and automation, enables implementation of actionable feedback loops for effectively managing and optimizing cloud-native infrastructure and applications. Target audience for this paper are application architects, SRE/operations leads, and business decisions makers, who want to better understand how observability can help with the transition of organizations and IT processes towards agile cloud operations.

During the last decade there has been a continued drive for increasing enterprise IT agility to support digitization of business processes and services. This trend has triggered a number of technology innovations like infrastructure virtualization and containerization. The flexibility offered by software automation has made it possible to create pipelines for staging rapid releases that include business logic and supporting platform components. With flexibility and speed comes complexity, which impacts how these new IT environments must be monitored and managed. With the increasing frequency of infrastructure and application changes it also becomes important to get timely feedback on the status of deployments and to quickly detect any significant infrastructure event(s).

The above paradigms are trends within the cloud native movement, and within this movement the "new monitoring" is referred to as *observability*. Observability expands the scope of traditional IT monitoring by applying a system-level view to telemetry data collection, covering metrics, events, logs, and traces. Intelligence is applied to telemetry data, producing an input feedback loop for controller and workflow automation, enabling automated changes and optimizations to infrastructure and application runtime deployments. In addition, telemetry data intelligence can be used for actionable reporting and analytics by organizational stakeholders.

IT agility is about deploying infrastructure and applications faster in a consistent secure, reliable, and repeatable way. To achieve this goal ongoing feedback is required to get insight in the state and health of applications and underlying IT infrastructure. Observability is focused on providing this feedback by collecting data from the IT technology layers. Real-time feedback and actionable information provides business benefits for both senior executive (CIO/CTO) and engineering staff in IT organizations. Observability in conjunction with Service Level Objectives (SLOs) can provide clarity and, this way, streamline problem resolution processes and reduce organizational friction. Leveraging observability, organizations can implement the required feedback loops and automation to effectively manage their evolving cloud-enabled IT infrastructure and keep up with the increasing agility of application deployments.

This paper presents a technology reference framework for implementing observability. The framework outlines the key functional technology areas that need to be considered for creating actionable feedback loops that use telemetry data for data center and cloud deployments. The technology framework can be used as guiding reference for evaluating and optimizing existing monitoring architectures and implementing new observability solutions.

Observability is an evolving technology trend that most likely will increase in importance due to the need for better monitoring visibility and control automation for distributed microservice applications and cloud infrastructure. In addition to cloud infrastructure and microservices, IT agility is also driving organizational change. Effective IT organizations require clear communication and delineation of responsibilities between infrastructure and application teams. SLOs can be defined to clearly articulate boundary of concerns between organizational stakeholders, providing tangible and measurable service quality levels. Leveraging observability, telemetry data and analytics can be used for monitoring and reporting on SLOs, providing transparency of IT service availability and clarity for streamlining IT operations.

Summary Overview of this Paper

This paper presents an overview of observability and discusses how observability is becoming a critical component for managing cloud-native IT environments, as part of the journey towards digital transformation. Cloud technologies, such as containers and microservices, enable application deployment agility but, at the same time, also introduce more technological and organizational complexity. Observability can help manage these complexities and provide actionable feedback loops, enabling better visibility and automation.

Telemetry Data and Feedback Loops

Observability is a discipline where intelligence is applied to different types of telemetry data, resulting into actionable inputs for automation and decision-level outputs for business and operations. Collection of telemetry data covers metrics, events, logs, and trace data. It is captured using agent instrumentation, SDKs, and web hooks. Observability expands the scope of traditional monitoring by applying intelligence to telemetry data to create a feedback loop for making control plane changes to cloud native infrastructure and applications, as shown in Figure 1. Depending on the complexity and impact of changes, actions can be implemented via autonomous controllers or manually by operations support. In addition, telemetry data intelligence can be used for planning and reporting.

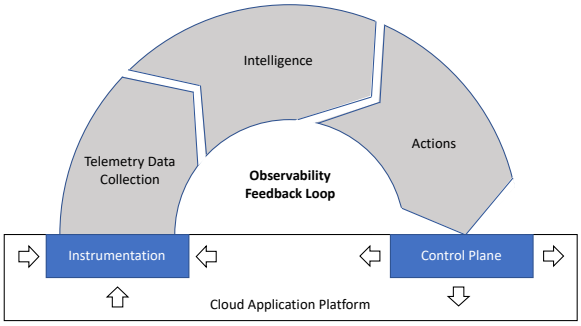

Figure 1 – Observability Feedback Loop for Making Control Plane Changes.

Cloud Observability Framework

This paper introduces a cloud observability framework for evaluating and implementing observability solutions, shown in Figure 2. The framework outlines the key functional areas that need to be considered for creating actionable feedback loops that leverage telemetry data for data center and cloud deployments. We formalize consumers of telemetry data as *personas* and use cases as *Service Level Objectives (SLOs)*.

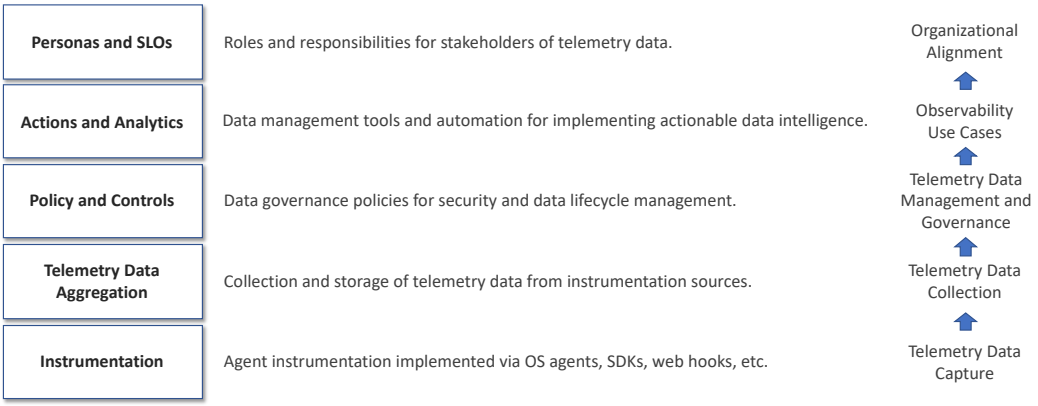

Figure 2 – Cloud Observability Framework.

Business Benefits

IT agility is about deploying infrastructure and applications faster in a consistent secure, reliable, and repeatable way. To achieve this goal ongoing feedback is required to get insights into the state and health of applications and underlying IT infrastructure. Observability is focused on providing this type of feedback by collecting telemetry data from the various technology layers in the IT stack. Real-time feedback and actionable information provide business benefits for multiple stakeholders, including senior executives (CIO/CTO), production support, and developers.

Organizational benefits:

- **Reduced organizational friction** – Minimize back-and-forth of emails and tickets between teams and organizations during trouble resolution via mutually agreed upon and monitored SLOs.
- **Streamlining of trouble resolution processes** – Access to centralized telemetry data and use of analytics for actionable insights, reducing the number of stakeholders required for resolving production issues.
- **Scaling of IT support via self-service capabilities** – Collected telemetry data can be made available to a wider group of stakeholders and end-users, enabling self-service monitoring and troubleshooting.

IT infrastructure benefits:

- **High availability of IT infrastructure and services** – Use observability intelligence to avoid and minimize problem resolution time of IT infrastructure and application deployment failures, reducing Mean Time to Detection (MTTD) and Mean Time to Resolution (MTTR).
- **Avoidance of capacity bottlenecks via proactive planning** – Use telemetry data trend analysis for proactive planning of usage capacity to prevent infrastructure congestion bottlenecks.
- **Optimize IT infrastructure capacity usage** – Track resource usage to minimize infrastructure overprovisioning and optimize infrastructure capacity usage to meet corporate sustainability targets.

Application benefits:

- **Acceleration of application development** – Telemetry data, such as application logs and metrics, help developers to fine-tune application runtime performance before production deployment.
- **Application performance monitoring** – Telemetry tracing data help application support teams to proactively monitor transactions and response times of application runtime components.

Current State of Observability

Observability is an evolving technology trend that will increase in importance due to the need for better monitoring visibility and control automation for distributed microservice applications and cloud infrastructure. This trend has industry traction and has become a key solution component for cloud native deployments. VMware's solutions for observability include Tanzu Observability (Wavefront) and vRealize Log Insight Cloud.

In addition to commercial observability solutions, open source software frameworks for implementing observability have become available as well. Some recent examples include graduated projects from the Cloud Native Computing Foundation (CNCF) focused on container monitoring and application tracing. Open source instrumentation, together with new observability metrics and tracing libraries in the Spring Boot Java development framework, facilitate the implementation of fine-grain telemetry data collection from cloud infrastructure and distributed applications.

Drivers for Observability

During the last decade there has been a continued drive for increasing enterprise IT agility to support digitization of business processes and services. This trend has triggered a number of technology innovations like infrastructure virtualization and containerization. These technologies were born out of the various paradigm shifts that were happening in the software development industry. Many were moving from monolithic architectures with few instances of platform components to more highly distributed implementations using microservices driving a higher feature velocity. Such flexibility offered by software-defined everything makes it possible to create pipelines for staging rapid releases that include business logic and the supporting platform components. With this high granularity, flexibility, and speed comes complexity. For example, new orchestration frameworks like Kubernetes and cloud infrastructure services have made it easier to provision new IT deployment environments and deploy application runtimes. At the same time, the underlying complexity has gone up significantly and this has become especially apparent in how these new deployment environments need to be monitored and managed.

With the increasing frequency of infrastructure and application deployments it becomes important to get timely feedback on the status of deployments and detect any significant related failure event(s). Deployment anomalies need to be detected and remediated quickly to minimize production environment impact. Experience has shown that the traditionally siloed eco system of IT monitoring tools is not adequate to handle the complexity and explosion of telemetry data associated with new containerized microservices application deployments. The above paradigms are trends within the cloud native movement, and within this movement the "new monitoring" is referred to as *Observability*. This new trend focuses on a system-oriented approach to IT monitoring, taking a holistic end-to-end view of monitored IT infrastructure by combining and processing different types of monitoring data feeds with a focus on generating actionable data as output. *Observability* has gotten industry traction and has become a key solution component for cloud native deployments.

This trend in IT monitoring is not much different than similar trends in other industry verticals like the airline and automobile industry. When systems got more complex, they resorted to more actionable monitoring that helped control the impacts of complexity in real-time. Flight engineers in cockpits of commercial airplanes, traditionally responsible for manually monitoring and operating the airplane's hydraulic and electrical systems, have been replaced by advanced computer systems that aggregate and process information from thousands of measurement sensors. Similarly, car manufactures replaced traditional dashboard gauges with centralized computer displays in new Electric Vehicles (EVs), not only for displaying sensor data but also for providing advanced driver assist and self-driving functions.

In short, the process of telemetry data collection for IT monitoring is now becoming a means to an end. The focus is shifting from just collecting (and observing) data towards processing heterogenous telemetry data in order to derive actionable information that can be used for value-added insights and automation, creating a 'driver assist' for IT.

Relationship Between Observability and Monitoring

The terms observability and monitoring are often used interchangeably but they have different meanings. Observability has its origin in control theory where it refers to the ability to derive the (internal) state of a system based on its external outputs. The goal is to externalize system state based on sensor data outputs. The concept has been made popular by cloud engineering and operations teams, typically focused on managing large distributed application deployments. Monitoring, on the other hand, originated in the operations and support world where it describes the discipline of collecting metrics and alerts to monitor health and performance of discrete IT infrastructure components (e.g., servers).

The observability principles of using a system-level view and leveraging multiple heterogenous data inputs are becoming key for effectively monitoring and managing cloud native deployments. Cloud technologies, such as

containers and microservices, enable application deployment agility but, at the same time, also introduce more technological and organizational complexity. Observability can help to address these complexities.

NOTE: Observability is a discipline where intelligence is applied to different types of telemetry data resulting in actionable inputs for automation and decision-level outputs for business and operations.

Observability expands the scope of traditional monitoring by applying intelligence to collected telemetry data to create actionable feedback loops for cloud infrastructure control plane changes. Traditional monitoring is mostly focused on visualizing collected telemetry data for manual observation.

NOTE: Observability expands the scope of monitoring, treating telemetry data as signals for deriving system-level status views of infrastructure and applications.

Using telemetry data intelligence and analytics, engineers and SREs can implement actionable feedback loops to make infrastructure and application runtime deployment changes. Figure 3 shows the different types of feedback loops that can be implemented using various levels of automation:

1. Manual actions: telemetry data reporting and (search) analytics used by infrastructure and application teams for troubleshooting and deployment planning.
2. Semi-automated actions: event policy rules and analytics that trigger workflow automation, such as pagers and tickets, which trigger manual change and troubleshooting activities by support teams.
3. Automated actions: telemetry data collected and processed by platform controllers to execute automated infrastructure changes, such as autoscaling and redeployment of containers.

For the first type of feedback loop advanced telemetry data analysis is used to support manual operations and planning activities, such as log search analytics and metrics timeseries trending analysis. There are a variety of data analytics

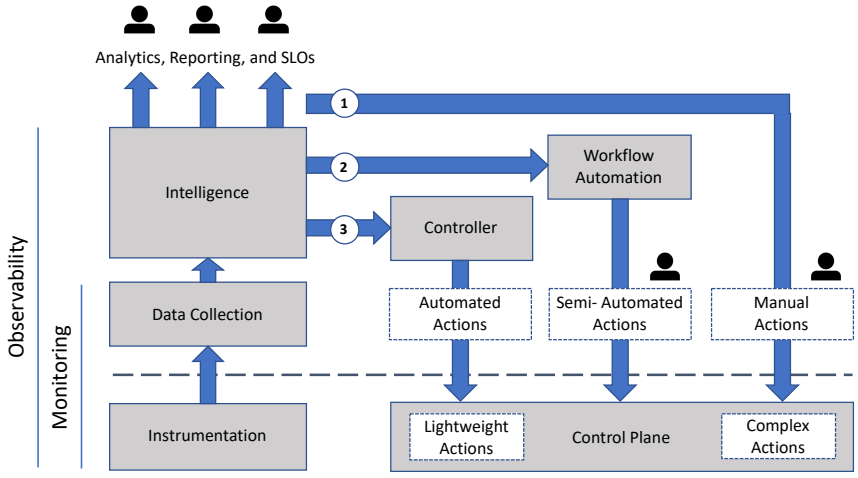

Figure 3 - Observability Feedback Loops and Monitoring.

tools and services available today for advanced analysis of telemetry data. In the monitoring space, use cases for these data management capabilities were initially used by full stack cloud engineers, especially for supporting

troubleshooting activities. With the increasing adoption of cloud-native infrastructure and distributed applications, the applicability and usage scope of telemetry data analytics is expanding. Examples include use of telemetry data for digital experience monitoring and business IT-level status reporting.

For the second type of feedback loop collected telemetry event data is being used for triggering automated incident management workflows, driving and coordinating manual operations activities. For example, a node failure triggers an exception event message, causing an automated trouble ticket to be created and assigned to a platform team for further investigation. Event messages can be generated by agent instrumentation (i.e., native events) or generated as part of processing of collected telemetry data (i.e., synthetic threshold alerts). New advanced data query and machine learning technologies complement existing alert processing functionality, such as contextual filtering and deduplication. Telemetry event data can now be processed more effectively and, as a result, optimize incident management workflows that are being triggered by alerting.

Finally, with the introduction of Service Mesh, a third type of observability feedback loop is becoming available. Telemetry data is being used as input for platform controllers that provide automation services (without any human intervention). For example, Service Mesh enables automated network service management for containerized applications.

NOTE: Leveraging observability, IT organizations can implement feedback loops and automation to effectively manage cloud-native infrastructure and keep up with the increasing agility of application deployments.

Addressing Organizational Complexities

Enterprise IT has traditionally been organized around teams and roles focused on infrastructure and applications. Infrastructure teams are focused on the deployment and operations of network, storage, and compute. Application teams are focused on development and deployment of business applications. This divide is primarily historical and the result of strict delineation between infrastructure operations and development teams in IT organizations. Nowadays, with businesses focusing on reducing cost and increasing deployment agility, frictions are starting to arise between application and infrastructure teams.

Areas of Responsibility and Visibility

One of the reasons for organizational friction is lack of visibility into the work and status of the different sides of the IT organization. Application teams often do not have visibility into the status and plans of IT infrastructure buildouts and upgrades. Infrastructure teams, in turn, often don't have visibility into application deployment schedules. Net result is that, when deployment failures occur, teams are surprised and multiple teams need to get involved for problem resolution, causing organizational overhead. High frequencies of production incidents and organizational IT silos often create a culture of finger pointing and escalation, instead of cooperation.

The delineation between responsibility and visibility for infrastructure and application teams is shown in Figure 4. Application teams are responsible for the development and deployment of applications. At the same time there is a need for these teams to get visibility into the underlying infrastructure. This visibility provides insights into application usage of infrastructure resources, which can be used for application performance tuning and infrastructure resource

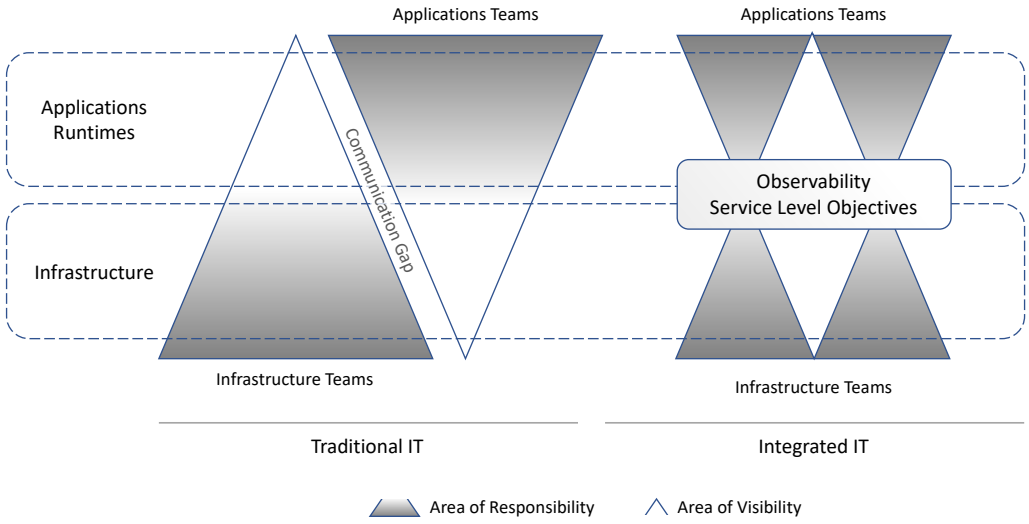

Figure 4 - Inverted User Requirements for Infrastructure and Application Monitoring.

usage optimization. In addition, visibility into underlying infrastructure health and failure events reduces the need for escalations (to infrastructure teams) and enables application teams to anticipate and minimize any application impacts.

Infrastructure teams are primarily focused on the deployment and health monitoring of the IT infrastructure, including virtualization and container orchestration platforms. Having better insight into planned application deployments helps infrastructure teams to anticipate and proactive manage aggregate infrastructure resource utilization by new application deployments. As a result, infrastructure capacity crunches can be avoided.

Observability provides holistic monitoring visibility for both infrastructure and application teams, reducing the communication gap and friction between teams and organizations.

With the adoption of cloud technologies, IT organization structures are starting to gradually change as well and new functional roles are being created that are starting to blur the lines between infrastructure, applications, and support responsibilities. An example of a new functional IT role is the *Application Platforms Architect*, as described in the OCTO Application Platforms Positioning white paper.

Site Reliability Engineering and DevOps

Site Reliability Engineering (SRE) is a concept that is often used as part of cloud transformation discussions. SRE is a discipline that originated out of the hyper scaler world. In short, it applies a software-development approach to system administration to perform system and service management activities more effectively. SRE leverages automation but is also focused on deployment optimization and troubleshooting.

SRE engineers spend only part of their time resolving production issues. The rest of their time is spent on automating manual tasks (via software development) and, this way, increasing the scale and effectiveness of IT operations. In addition to manual tasks, SREs also codify rules of engagement, which are implemented via Service Level Indicators (SLIs), Service Level Objectives (SLOs), and Service Level Agreements (SLAs). Using observability and leveraging telemetry data, SRE engineers can monitor system-level compliance of these agreed upon performance indicators.

Although there has been a lot of industry mindshare, not many mainstream enterprises have fully adopt and implemented SRE principles across their entire organization. There are multiple reasons for this including lack of engineering talent, (complex) multi-vendor product architectures, and organizational inertia. SRE team members are

multi-talented individuals that need to possess both software developer and system administration skills. This skillset is hard to find and in high demand. In addition, contrary to the highly customized and optimized software platforms developed by the hyper scalers, typical enterprise IT environments cover a multitude of products and technologies, limiting the implementation of custom SRE-like monitoring and automation.

Despite these challenges, enterprises are starting to gradually acquire skillsets and concepts from the SRE world. SRE remediation and monitoring activities leverage observability for collecting actionable information. This approach can be a first step for implementing SRE practices in an enterprise IT organization. Observability can provide insights and tooling intelligence for IT staff to work more effectively and reach across technology domains and organizations. Finally, with the adoption of cloud and Kubernetes container platform technologies, enterprises now have an option to leverage advanced platform automation capabilities such as Hybrid Cloud Runtime (HCR), which enables automation of SRE-type functions and, this way, help with alleviating the SRE skillset challenge.

Service Level Objectives and Personas

It will take time for mainstream enterprises to broadly adopt a nimble and integrated IT organization model. *Service Level Objectives (SLOs)*, however, are a concept out of the SRE and DevOps world that can be implemented as a first step towards organizational agility and this is also where observability comes in. SLOs can be leveraged to establish mutually understood and agreed upon service availability and quality objectives for specific IT infrastructure services and applications, discussed further in the OCTO blog SLOs – The Emerging Universal Language in the Enterprise. By leveraging observability, telemetry data can be used for monitoring *Service Level Indicators (SLIs)* to validate whether mutually agreed upon SLOs are being met. Observability, in this case, offers a methodology for monitoring compliance of mutually agreed upon SLOs between teams and organizations.

NOTE: With the evolution towards cloud-native technologies and increasing speed and agility of application deployments, organizational roles and structures need to evolve as well, requiring changes in IT operations and application development.

With the evolving organizational roles and responsibilities, an increasing number of stakeholders is starting to leverage telemetry data intelligence to help with day-to-day job responsibilities. For example, application developers use application logs and tracing data to debug application code and application architects use application end-to-end delay tracing and infrastructure utilization metrics to optimize application architecture designs and deployment sizing. These are examples of observability *personas* leveraging telemetry data. A *persona* is an abstract representation of a specific type of stakeholder.

Cloud agility in IT organizations requires streamlining of roles and responsibilities, interactions, and operational processes. Formalizing organizational roles and responsibilities using *personas* can help with clarifying and delineating areas of focus and concerns. For each persona, specific requirements can be defined for usage of telemetry data to perform specific tasks. In addition, SLOs can establish measurable application and infrastructure service attributes, streamlining communications between personas. By leveraging observability feedback loops, telemetry data can be used for monitoring and reporting on SLOs.

Observability for IT Organizations

Migration towards the cloud impacts technology, people, and processes. Figure 5 illustrates this transition at various levels across the IT organization and technology stack from a monitoring and observability perspective.

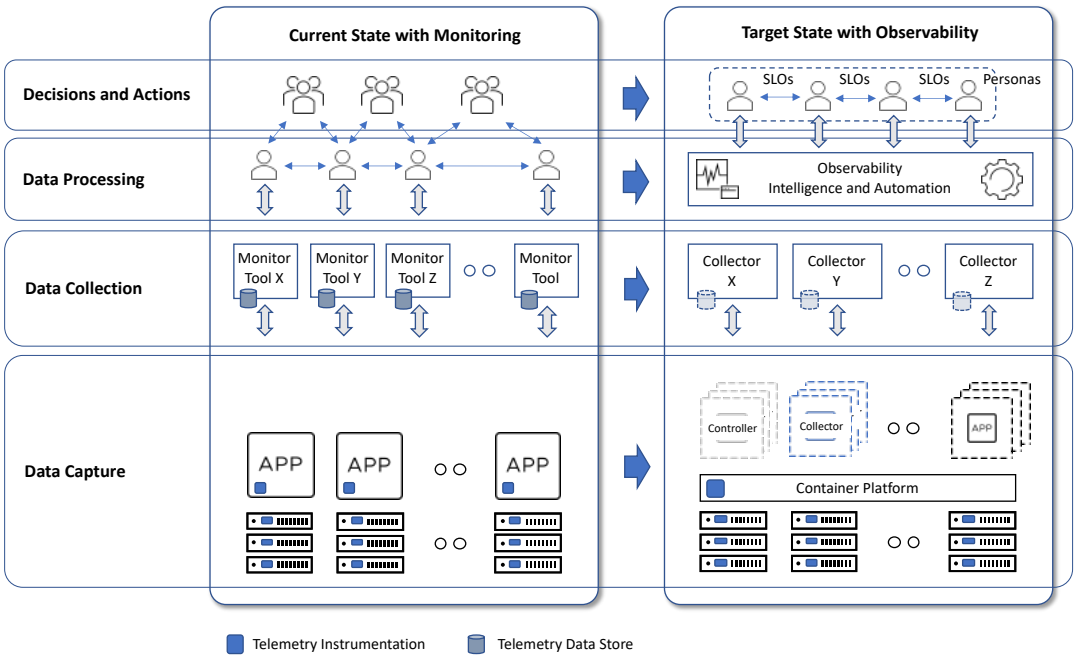

Figure 5 – Transition from Monitoring Towards Observability.

Domain-specific monitoring in traditional IT environments often results in telemetry data silos, with data being used for manual observation only by select users. As a result, planning and decision making in IT organizations is typically hampered by lack of information. Access to information requires coordination with multiple teams, causing complexity and delays. With observability, integrated telemetry data flows and feedback loops can provide holistic visibility into the state and performance of the IT ecosystem. In addition, data analytics and automation are used to streamline and automate the use of telemetry data. Mutually agreed upon performance targets are defined via SLOs, which are being monitored using telemetry data. This facilitates the streamlining of interactions between application and infrastructure teams, enabling IT organization agility.

NOTE: Cloud adoption and observability are driving a data-driven organizational culture where telemetry data is being used as integral feedback mechanism to facilitate status monitoring, trouble shooting, and deployment optimizations.

The changes across the layers in the IT ecosystem, covering organization and technologies, are shown in Figure 5 and can be summarized as follows:

- **Decisions and actions** – Well-defined interactions and communication using SLOs between users with multi-disciplinary responsibilities, replace ad-hoc escalations and interactions between organizations.
- **Data processing** – Automated intelligence, used for getting actionable insights from telemetry data, replace siloed monitoring done manually by multiple operators independently in different organizations.
- **Data collection** – Scale-out configurable data collectors, aggregating telemetry data into centralized datastore(s), replace standalone monitoring tooling with local siloed datastore(s) for telemetry data.
- **Data capture** – Configurable telemetry software embedded in application code and across various layers in the IT stack, enhance vendor- and domain-specific agent and code instrumentation.

Operational changes that result from the migration towards cloud infrastructure and applications impact both deployment of instrumentation, and collection and usage of telemetry data. With distributed applications being deployed on multi-layered cloud infrastructure, more granular telemetry data needs to be collected to understand the state and dependencies between multiple technology layers and distributed applications. As a result, telemetry instrumentation needs to be deployed at various layers of the IT stack. Instrumentation can no longer be an afterthought and must become an integral part of deployment automation pipelines. Examples include automated telemetry instrumentation deployments as part of server OS and Kubernetes platform builds, and application runtime deployments. Because the volume, scope, and uses of telemetry data are rapidly increasing, more emphasis on collection and storage of this data is required. Ingest of large volumes of telemetry data requires careful design and fine-tuning of tools and storage repositories to prevent data ingest performance delays and storage scale limits. Traditional monitoring tooling may have to be replaced or complemented with additional data management capabilities, expanding the scope of existing monitoring deployments to align with Big Data system architectures with data ingest pipelines and centralized data lakes.

The richness and increasing volume of telemetry data makes manual analysis of this data increasingly challenging. To derive actionable insights (search) analytics tools are needed. Some of the time spent on manual data analysis will be replaced with implementation of data query rules and automation, as part of ongoing operational activities. In effect, operations support engineers are starting to take on a data analyst role.

Cloud Observability Framework

Figure 2 (above) presents the cloud observability framework. An expanded version of this framework is shown in Figure 6, detailing specific functions for each layer and illustrating an integrated and distributed implementation of these functions.

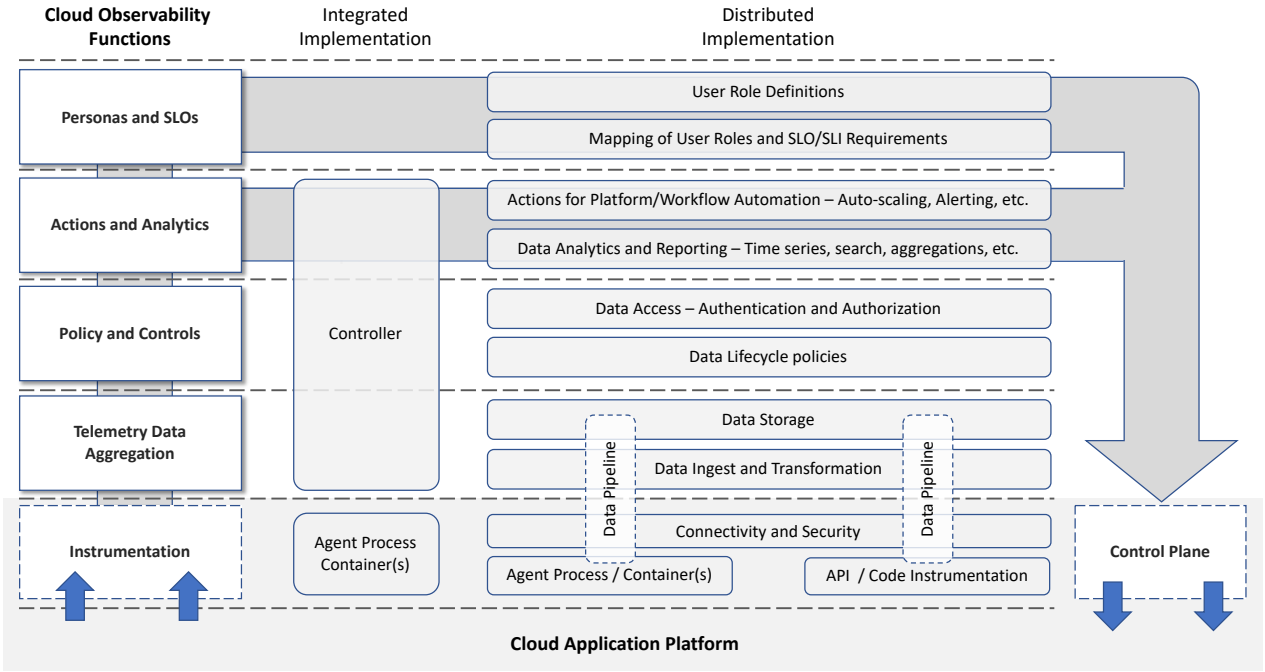

Figure 6 – Cloud Observability Framework – Expanded View.

The cloud observability framework can be used as reference for evaluating existing monitoring architectures and implementing new observability solutions using monitoring tools, cloud monitoring services, or a combination of both. Each function in the framework can be considered as separate focus area for design and implementation decisions.

The functional building blocks of the cloud observability framework can be summarized as follows:

- Personas and SLOs – Personas and SLOs can help align and streamline the use of telemetry data among organizational stakeholders. Personas define specific roles and responsibilities in an IT organization. SLOs can be used as blueprint for defining mutually agreed upon quality levels for specific IT services and applications that personas care about. To monitor SLOs, SLIs need to be identified. SLIs can be translated into specific metrics that can be collected, calculated, and reported on over time.
- Actions and Analytics – In several observability use cases telemetry data is used to support operational functions such as reactive trouble shooting, proactive planning and trending analysis, SLO status reporting, and platform and workflow automation. To make collected telemetry data useful and actionable to support these use cases, aggregation and processing of large data sets is often needed, requiring specialized (big) data analytics tools and search capabilities.
- Data Policy and Controls – If telemetry data storage is required, data governance policies, covering data security and lifecycle management, need to be considered. Security controls must be in place to ensure authenticated access with authorized scope of telemetry data for specific personas, which can be implemented as part of an enterprise identity management system. Data access policies are especially important for multi-tenant deployment environments. In addition, data retention policies need to be defined for specific telemetry data types to prevent unwieldy storage capacity usage. Retention policies are especially important for managing the explosive growth of telemetry data generated by containerized microservices. Different data retention intervals are often defined for telemetry data collected from different deployment environments (e.g., development, test, and production).
- Telemetry Data Aggregation – Telemetry data is collected from instrumentation endpoints. The end-to-end process of collecting telemetry data covers connectivity and security (session authentication and encryption), data ingest processing (e.g., field transformations) and data storage. Connectivity and storage implementations can be influenced by business continuity requirements, dictating not only storage resiliency but also transport resiliency via live-live (dual) data collection feeds, for example. In addition, deployment guidelines for cloud-based monitoring services may dictate additional security requirements for telemetry data traffic sent to and from external cloud service endpoints.
- Instrumentation – Agent instrumentation is deployed for capturing telemetry data from the various layers in the technology stack. With the evolution towards cloud native deployments, methods for generating and capturing telemetry data are expanding beyond (well-known) OS agents and now include containerized metrics collectors and log forwarders, proxy forwarding agents, application runtime libraries, web hooks, and service APIs.

Not all functions in the cloud observability framework are always applicable for implementing observability feedback loops, depicted by the integrated implementation in Figure 6. For example, a ServiceMesh controller in a Kubernetes cluster can collect and process telemetry data locally without the need for storing data or for human interaction. Personas and data lifecycle policies may not be applicable in this scenario. Integrated implementations combine multiple functional building blocks of the framework into one or more execution runtimes.

Observability feedback loops can also be implemented using a distributed architecture with multiple systems and services supporting specific functions defined in the cloud observability framework. This type of architecture can be considered a distributed observability implementation where, in addition to telemetry data collection, intelligence is deployed to enable actionable workflow automation and data analytics for end-users.

To determine the scope and projected volume of telemetry data it is useful to first consider the usage and users of this data. Personas, SLOs, and use case definitions can be used as guideline for fine-tuning the scope and type of telemetry data that should be collected. This way, a common pitfall in IT organizations can be avoided where a lot of unnecessary telemetry data is being collected that is never used and useful telemetry data is collected that is never made available to the users who need the data.

Telemetry Data Pipelines

The collection process for specific types of telemetry data can be organized into separate data pipelines. For example, separate data pipelines can be in place for collection of metrics, events, and logs. Data pipelines enable grouping and organizing of functionality for collection, ingest, and storage of specific telemetry data types. In addition, functional data pipeline requirements can be used for evaluating telemetry data collection tooling and technologies. It is not uncommon to have dedicated tooling and services for support of each telemetry data pipelines.

Instrumentation for Observability

The phased evolution of applications and underlying infrastructure is illustrated in Figure 7. Monolithic application runtimes are broken up into containers, supporting microservices. Single-server OS deployments are evolving into a multi-layered application platform. Along with these changes, the instrumentation footprint and scope of telemetry data capture across the functional platform layers is changing as well.

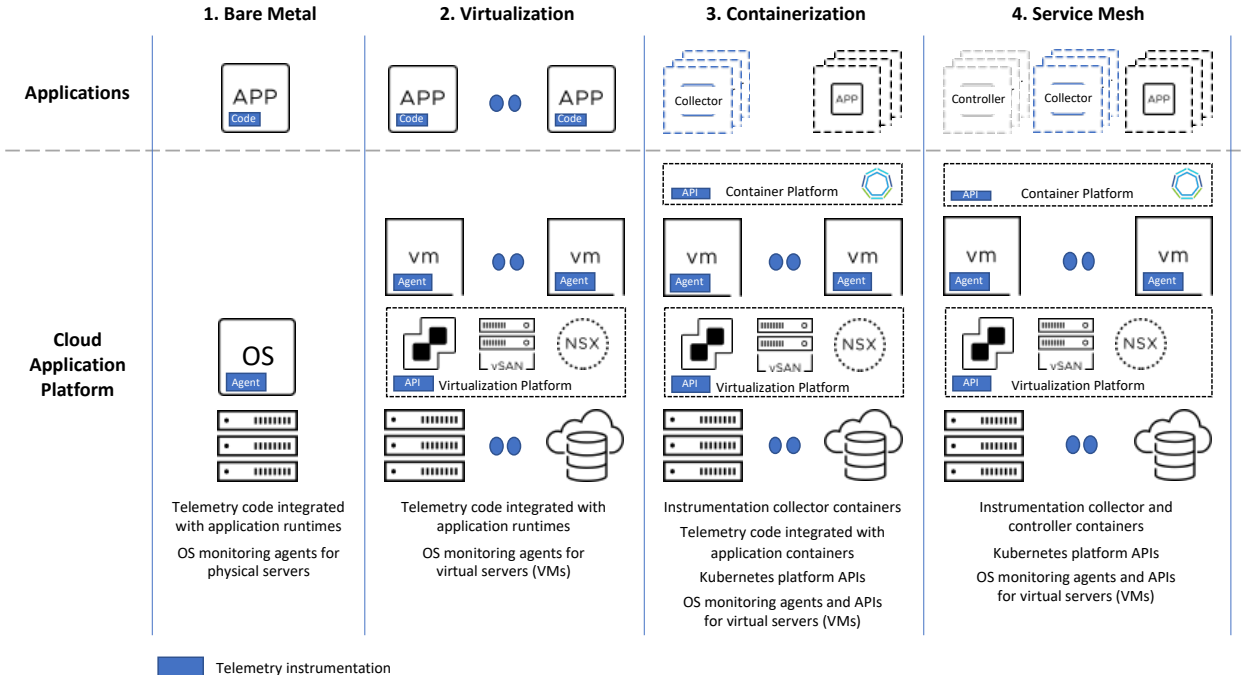

Figure 7 - Instrumentation Evolution for Observability.

To expand the full scope of monitoring across the functional application platform layers, OS agents are complemented by agent instrumentation deployed as part of the virtualization and container platform layer.

With monolithic application runtimes evolving into containerized microservices, the scope and footprint of embedded (APM) agent code instrumentation to collect application-level health and performance data is changing as well. Embedded instrumentation application code is complemented, and in certain cases replaced, by side car container

monitoring services, providing additional visibility into the performance and health of application container deployments. An example of this type of service is ServiceMesh, which offers a new distributed monitoring and control service framework for containerized applications.

With the release of new open source monitoring software tooling, more options to deploy custom instrumentation for capturing specific application and platform-level health and performance. At the same time, increasing options for capturing telemetry data also introduce the potential for more deployment complexity, which needs to be managed carefully. Making installation of monitoring instrumentation part of platform deployment automation ensures consistent monitoring visibility of deployed infrastructure platform resources. However, if ongoing changes are required to update data collection policies, such as log parsing rules, separate deployment pipelines for these instrumentation runtime components may be required.

Types of Telemetry Data

Different types of telemetry data can be collected for observability of applications and the underlying cloud application platform. In general, there are two methodologies for collecting telemetry data. The first one uses defined time intervals for collecting telemetry data with a focus on obtaining (point-in-time) instance values of known resource or state (counter) variables. CPU and memory utilization are examples of these type of variables. This type of telemetry data is especially useful for *planned* and forward-looking activities, such as establishing baselines and performing (retroactive) trending analysis. The second methodology is focused on collecting ad hoc telemetry data, which can be generated by a source system or process at any point of time. The content and timing of this data in not known ahead of time. This type of telemetry data is mostly used for *unplanned* remediation activities, where the focus is on reactive troubleshooting and root cause analysis.

The following types of telemetry data are commonly used for implementing observability:

- Metrics – Measurement of some state or counter value at a point in time. Metric values measured with some (sampling) frequency over a period of time form a data time series, which can be used for advanced analysis such as histograms for analyzing one or more metric distributions over time.
- Events – System-generated messages, typically sent as the result of a state change or error condition. Alerts are a special type of event, indicating some failure condition. Events can be generated at any point in time and the message format is typically short with a specific syntax.
- Logs - System and software runtime-generated messages, typically generated on an ongoing basis to record system change or software runtime state output. Logs can be generated at any point in time and are typically more verbose with variable syntax. For system OS log messages the level of granularity is configurable using specific logging (debug) levels. In addition, developers can codify custom log messages as part of application runtime outputs.
- Traces – Information about service-to-service calls, collected via aggregation of span workflow messages, which are sent asynchronously by instrumented application components that participate in the application transaction path. Individual span messages can also include log information (span logs).

Metrics and traces are examples of telemetry data collected on an ongoing basis where the focus is on obtaining point of time values of known system variables and characteristics over a period of time. Events and logs are examples of ad-hoc telemetry data that is being collected as part of ongoing monitoring.

Metadata

Traditional system monitoring of (bare metal) server deployments typically involves collection of metrics, events, and logs. In this type of deployment, a single application runtime is deployed on a server and all telemetry data collected from that server can be associated with the application runtime and underlying server system. There is a one-to-one topology relationship between the application and the underlying host system. In cloud native deployment environments with microservices this relationship is more complicated. New scale-out architectures enable distributed applications to be deployed in multi-host cluster configurations. In addition, new microservice architectures enable applications to be deployed as separate container runtimes that run on multiple host systems for increased scale and resiliency. Metadata can be used for keeping track of the multiple runtimes associated with a given application. For example, in Kubernetes label selector meta data can be used to assign semantics to specific PoDs, associating containers with application tiers. L the Kubernetes platform layer will also detect and alert on failure events abel selectors are also being used extensively by container management services, such as the Kubernetes scheduler.

Meta data is becoming increasingly important for referencing and processing telemetry data. As part of the data collection process, meta data can provide additional context for further processing of telemetry data. For example, meta data tags can be inserted as part of custom application metrics and tracing code, enabling analytics tools to organize and infer topology relationships from large volumes of this telemetry data. Figure 8 show an example of additional context that can be associated with cloud application platform resources using meta data fields.

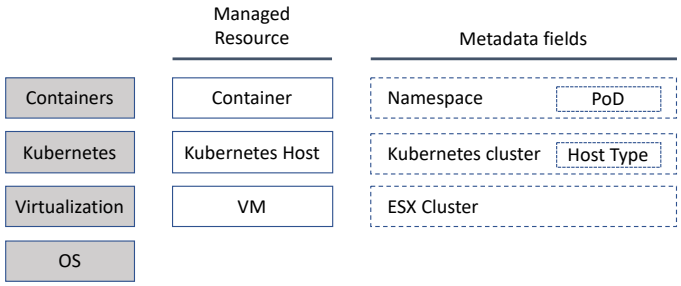

Figure 8 - Use of Metadata Fields for Additional Context.

Metadata can be added at various stages of the telemetry data collection process. For example, metadata can be added by agent instrumentation, as part of the post processing phase before telemetry data is sent upstream to an monitoring aggregation system or service. In addition, metadata can be added as part of ingest processing by data collection systems and cloud monitoring services. This is typically the case with log ingest pipelines where context-specific metadata is added to log messages, as part of log field tokenization processing.

Observability Data Types

Figure 9 shows various types of telemetry data as observability data type. Metrics, traces, logs, and events can be considered separate implementations of this abstract data type, each characterized by specific syntax and temporal characteristics. Meta data is the common attribute that, in addition to providing additional context, enables grouping and association between different observability data types.

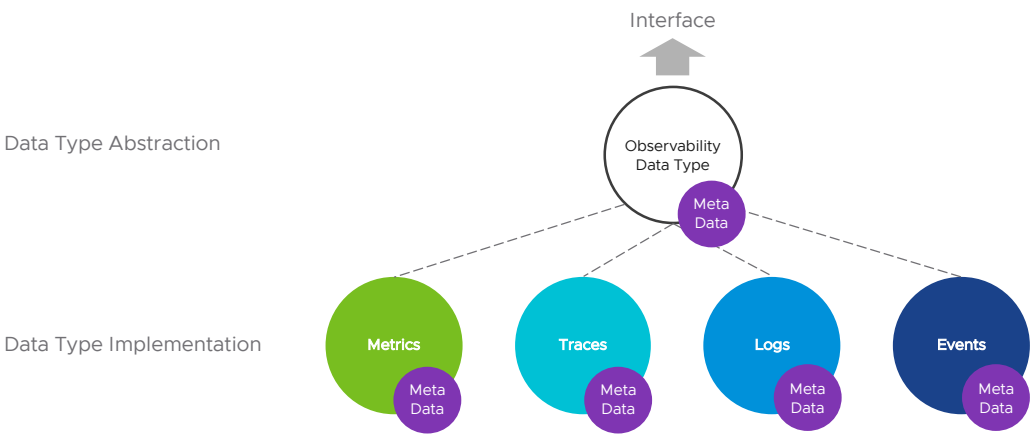

Figure 9 – Observability Data Types.

Events Versus Logs

The distinction between events and logs can be blurry in certain scenarios. For example, during production outages multiple failure events and error logs are being generated. Failure event (alert) messages can be used for triggering an incident management workflow, such as auto-creation of support tickets and Slack notifications. At the same time, error logs can be used for post-mortem analysis of system and application failure behavior. In theory, error log messages could also be used for monitoring failure events (and triggering incident workflow). This approach, however, can have undesirable side effects. Enabling verbose infrastructure and application code logging can quickly lead to data storage and throughput overload, potentially impacting production systems. In addition, logging volumes tend to grow in tandem with the growth of applications and infrastructure deployments, which eventually become unmanageable. Finally, transport delivery of log messages is not always reliable and error log messages potentially can get lost. The recommended approach for failure detecting is to use event collection and tracing where possible and limit logging in production environments. Logging is a useful tool for application and infrastructure debugging and post-mortem failure analysis. Log collection is therefore mostly applicable for development and test environments[1].

Topology and Inventory Data

In addition to telemetry data, such as metrics and logs, topology and inventory data is also being collected as part of monitoring in traditional production environments. Topology information uniquely identifies and derives parent-child relationships between deployed infrastructure and application resources. For example, the relationship between a host system and the application processes running on this system. Topology information is especially useful for failure alert processing and root cause analysis. Leveraging topology inheritance relationship data and filtering rules, alert message streams can be pruned in order to pin-point root cause alert(s) that can be analyzed further. Collected inventory data about physical, virtual, and logical infrastructure and about application runtimes is often used for inventory compliance tracking and license management.

Topology data is usually collected from a variety of data sources, leveraging different collection methods such as discovery agents and ping sweeps across deployment environments. Collected inventory and topology information

[1] Leveraging search analytics, it can be useful to scan production application logs for any debug messages, as a way to clean up application code and prune log output. Don't be surprised the first time you see the results.

has traditionally been stored in a centralized Configuration Management Database (CMDB). With the high frequency of changes and scale in cloud-native deployment environments, this centralized storage model is becoming unattainable and must be changed.

Discovery methods for cloud-native infrastructure and applications are changing and, instead of scheduled discovery tasks, on-demand queries and service discovery APIs are being used to obtain the current topology and inventory information. In addition, via metadata, source topology relationships can be derived from collected telemetry data. Net result is that domain-specific topology information is now typically maintained by multiple systems and services and accessible on an on-demand basis, instead of being stored statically in a centralized datastore.

Instrumentation Deployment Considerations

Deployment of instrumentation to capture telemetry data at various technology layers requires careful consideration. Tradeoffs have to be made between the required level of monitoring visibility, resource performance impact, and deployment complexity. That is, a more granular level of instrumentation can provide a deeper level of monitoring visibility but also adversely impact system performance.

For infrastructure monitoring in large complex IT environments it is typical to see dozens of OS agents installed on production server nodes. With each OS agent taking up a slice of system memory, the net effect can be that total aggregate system memory consumed by all these OS agents severely impacts overall system performance. This is commonly referred to as 'agent bloat'. In addition to server compute and memory overhead, maintaining multiple software agents on the same OS creates operational complexities, especially for agent software installs, upgrades, and patching. In this scenario, multiple teams may need to get involved in maintaining software agents on the same server OS, requiring time consuming coordination and creating deployment risks. In addition, there are scenarios where specific agent instrumentation code consumes large amounts of system resources, impacting overall system performance. For example, it is not uncommon for certain third-party monitoring agents to consume GBs of system memory. This problem can get exacerbated in application container runtime deployments, where memory overhead of embedded agent instrumentation can get replicated with each new container deployment.

New open source software solutions for implementing and deploying agent instrumentation are becoming available, providing more flexibility and scale for capturing telemetry data. For infrastructure and OS monitoring new open source frameworks, such as Prometheus and Fluentd, enable configuration of custom metrics and event checks and, this way and provide an opportunity to consolidate and optimize the footprint of agent instrumentation. Similarly, for application runtime monitoring open source instrumentation libraries have become available, enabling custom and more seamless integration of span tracing output routines in application code. Examples are the Sleuth and Wavefront libraries in the Spring application development framework, which enable distributed tracing capabilities and integration with the Tanzu Observability by Wavefront service. Finally, for scenarios were tracing instrumentation routines cannot be integrated into application code, distributed application delivery controllers, such as VMware's Avi Networks, can be used to analyze application transaction traffic without any application code changes.

Observability Use Cases

The example deployment scenario shown in Figure 10 illustrates how the cloud observability framework, presented in Figure 6, can be applied to cloud-native IT deployments. In this example scenario, three multi-tenant Kubernetes clusters are deployed, leveraging nine VM compute nodes. In addition, three containerized applications are deployed; App X, Y, and Z. Separate data pipelines are in place to collect metrics, logs, and events, which are aggregated and curated for use by various organizational stakeholders, reflected by different personas. Instrumentation is deployed across various technology layers in order to collect telemetry data, covering the VMware SDDC, Kubernetes platform, and application container layer.

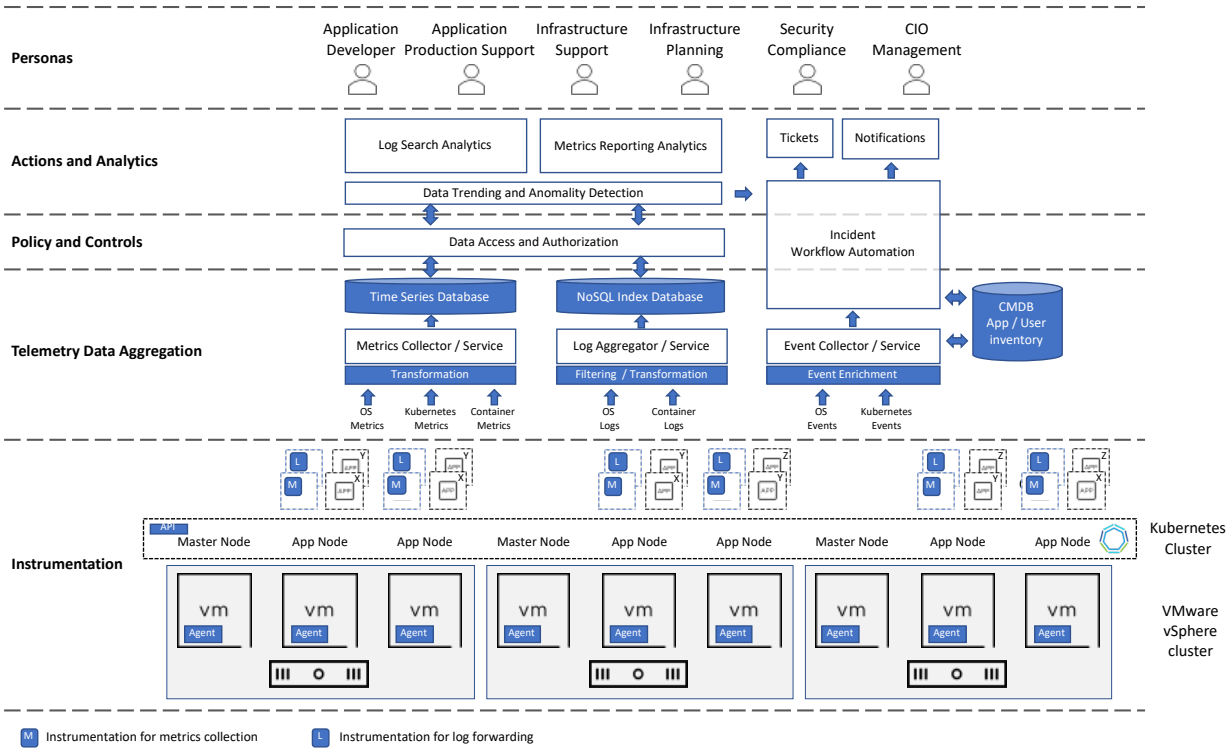

Figure 10 - Observability Deployment Example.

For simplicity, this deployment scenario does not include monitoring of network and storage infrastructure, application CI/CD deployment pipelines, and operational health status of monitoring tooling themselves. Also, in real-world scenarios dedicated infrastructure is typically allocated for support of dedicated deployment environments, covering development, test, and production.

Personas

In large enterprise IT organizations multiple stakeholders for support of the delivery and deployment of business applications can be identified. In the deployment scenario in Figure 10 multiple personas are defined, reflecting some of these different stakeholder roles and responsibilities.

The *Application Developer* persona represents the members of the three application development teams (for App X, Y, Z, respectively).

There is one shared application production support team, responsible for the deployment of new application code releases and ongoing monitoring of application health and performance. Members of this team, represented by the *Application Production Support* persona, work closely together with the application development teams.

Infrastructure Support personas handle day-to-day support of VMware vSphere clusters plus associated networking and storage infrastructure, and the buildout and maintenance of Kubernetes clusters. *Infrastructure planning* personas focus on ongoing monitoring of cluster capacity usage and are responsible for planning and buildout of new cluster capacity. Members of the security team, *Security Compliance* personas, are focused on detecting any infrastructure and application-level security threats. Finally, the *CIO Management* persona represents senior IT management staff and business decision makers.

Table 1 – Observability Use Cases by Persona.

Persona	Observability Use Case(s)	Telemetry Data
Application Developer	• Application code debugging • Application code performance optimizations	• Application container logs • Application metrics
Application Production Support	• Application runtime status and performance monitoring	• Application container metrics and events • Application container logs
Infrastructure Support	• vSphere cluster status monitoring • Kubernetes cluster status monitoring	• VM metrics, logs, and events • Kubernetes metrics and events
Infrastructure Planning	• vSphere cluster capacity monitoring • Kubernetes cluster and namespace capacity monitoring	• VM metrics and events • Kubernetes namespace and PoDs metrics
Security Compliance	• OS and container security monitoring	• VM OS logs • Kubernetes security logs
CIO Management	• Reporting on deployed applications and runtime characteristics	• Kubernetes namespace and PoDs metrics

Table 2 – Example SLOs and SLIs defined for Logging and Kubernetes Platform Services.

SLO	SLO Criteria	Measured SLIs
Application Container Logging Services	Availability: • Logs will be available (in service UI) within 5 minutes after being collected from application container • Log throughput can't exceed 500MB/minute • Log data retention is 4 weeks Exception handling (SLA): • Log ingest delays and peak throughput result into a ticket being created for app support team for awareness and follow up • Application teams can raise ticket to report service issues to the app support team with 1 hour response time for severity 1 issues • Parsing errors as result of log syntax errors may result in log message loss and ingest delays Pre-requisites (SLA): • Message format of ingested logs must conform with (nested) JSON syntax	• Container log ingest queue depth (at ingest backend) • End-to-end log processing time; average time difference between log generation and log storage in backend system • 95 percentile measurements for per-minute log ingest volumes
Kubernetes Platform Services	Availability: • 99.9% uptime availability of Kubernetes service • 99.99% uptime availability of Kubernetes cluster nodes Exception handling (SLA): • Kubernetes cluster failure result into a ticket being created for app support team for awareness • App support team can raise ticket to report Kubernetes service issues to platform support team with 10 minute response time for severity 1 issues • App support team is notified about Kubernetes cluster capacity threshold alerts, which may impact cluster availability	• Kubernetes failure alerts • Kubernetes node OS (threshold) alerts, covering disk and memory resources

Depending on the role, users (personas) have needs for specific information, which is captured by the various observability use cases defined in Table 1. *Developers and production support* require detailed telemetry data for debugging and troubleshooting, whereas *CIO management* needs more summarized information to track overall scope and state of production deployments. For example, summary deployment metrics such as number of application deployments, average number of pods and containers used by applications, average application deployment utilization, and number of escalated failure events during a given time period.

Key Performance Indicators and Service Level Objectives

In order to streamline communications between the application developer and application support team an SLO is defined for application container logging services. Similarly, an SLO is established for the Kubernetes services, which are deployed by the infrastructure team and used by the application support team. Table 2 summarizes the SLO details used as benchmark for SLIs, which are measured and reported on using collected telemetry data.

In addition to SLO targets, procedures for dealing with exception scenarios are included in Table 2 as well. Note that in many cases for measurement of SLIs data analytics need to be applied to time series of telemetry data samples. For example, 95th percentile analysis of time series of measurement points and time span analysis of received failure alerts.

Metrics Collection

The metrics data collection pipeline in the example deployment is illustrated Figure 11. It shows multiple users and use cases of resource metrics information, covering the various functional layers in the IT stack. *Application Production Support* is leveraging resource metrics for monitoring application container state and utilization. *Infrastructure Support* is focused on Kubernetes platform and VM node monitoring and is using resource metrics, such as memory and disk usage, to monitor platform status and trouble shoot production issues. Finally, *Infrastructure Planning* is leveraging metrics for trending analysis of VM node OS resources and Kubernetes platform resource usage, including comparing actual utilization against allocated namespace resource quotas.

Resource metrics are collected using OS and containerized agent instrumentation, covering OS and container-level CPU, memory, and disk usage metrics. In addition, Kubernetes provisioned resource metrics are collected, covering resource quotas and limits. As part of the collection and aggregation process, Kubernetes-level meta data is added to metrics to provide additional context, such as cluster node type, namespace and PoD identifiers. Metrics data is collected and stored in a centralized time series data store, which can be an on-prem deployment or a cloud-based metric collection service, such as Tanzu Observability by Wavefront. Data lifecycle policies are configured such that metric data is kept for a specific period of time and data governance policies are configured to ensure select access to application namespace and container metrics data for *Application Production Support*.

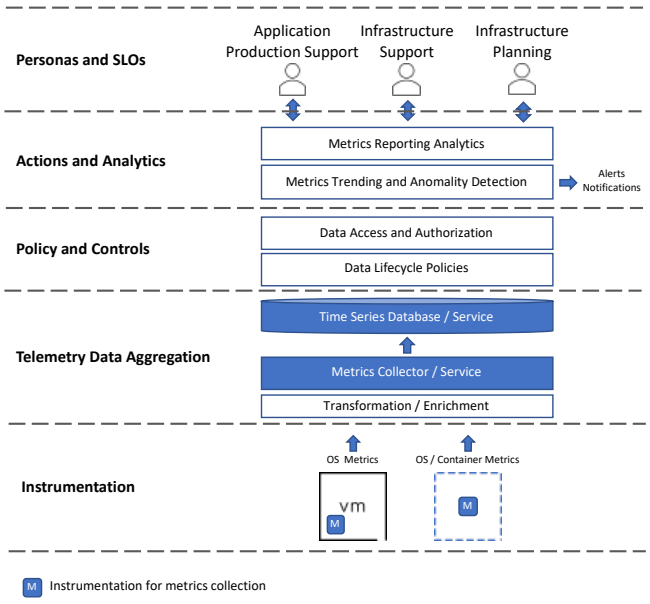

Figure 11 – Metrics Data Pipeline Example.

Data search, trend analysis, and aggregation analytics provide actionable insights into collected metric data, which can be used for trouble shooting and planning

activities. Aggregated resource metrics are used by *Infrastructure Planning* to derive the current utilization and availability of VMware vSphere clusters. Based on time series trend analysis, projected available and required cluster compute capacity is being calculated, which drives infrastructure capital investment for new cluster buildouts. The *Application Production Support* team is using aggregated resource metrics to track Kubernetes namespace compute and memory resource availability against set resource quotas for each application team. In addition, container-level resource metrics are being used to troubleshoot PoD health and performance issues.

Log Aggregation

Figure 12 shows the data pipeline for collecting log telemetry data from various functional layers in the cloud infrastructure. Multiple personas and associated use cases for log information can be identified, ranging from application, infrastructure, and security compliance. *Application Developers* leverage application container logs for trouble shooting and optimizing application code. *Application Production Support* uses application container logs for trouble shooting containerized application runtime deployments. VM OS and Kubernetes logs are used by *Infrastructure Support* for Kubernetes cluster and node troubleshooting. Finally, *Security and Compliance* monitors VM OS and Kubernetes audit security logs and VM Power ON/OFF and vMotion log events for tracking licensed software server instances.

OS and containerized log forwarders are being used to collect and process local logging output for OS, Kubernetes platform, and containers. This log data, in turn, is forwarded to a centralized log aggregator where log field transformation and filtering takes place as part of the log ingest process. Logs are stored in a specialized NoSQL data store, which can be data center deployment or a cloud-based log aggregation service, such as vRealize LogInsight. Similar to the metrics collection process, Kubernetes-level meta data is added to log messages providing additional context, such as cluster node type, namespace, and PoD identifiers. Data lifecycle policies are configured such that log data is kept for a select period of time, which is especially important due to the potential for high log data volumes. Data governance policies ensure select access to application container logs for *Application Developer* and *Application Production Support* roles and restricted access to audit and security logs for *Security and Compliance* team members. With multiple application teams sharing the same Kubernetes cluster infrastructure, multi-tenant access and authorization policies are implemented to ensure that each developer team only has visibility to log information associated with their respective application containers. In addition, OS and Kubernetes related logs are only accessible to *Production Support*.

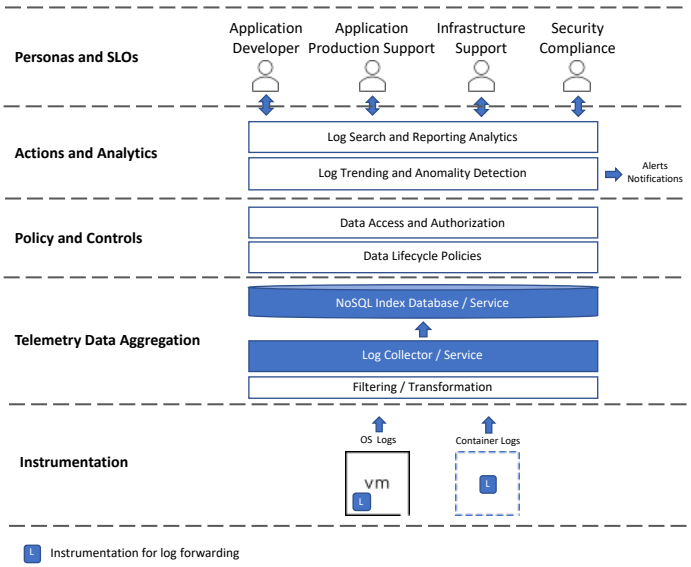

Figure 12 – Log Data Pipeline Example.

Search and data aggregation analytics are used for analyzing log data, especially for trouble shooting, for example, for isolating log patterns based on specific key words in order to pinpoint specific failure conditions. *Application Developers* use logs for application code debugging and *Application Production Support* use container logs for trouble shooting application container runtime deployments. OS and platform logs are used by *Infrastructure Support* for troubleshooting of Kubernetes node and platform failures.

Alerting and Incident Management

The event data collection pipeline in the example deployment is shown in Figure 11. Event and alert monitoring primarily apply to infrastructure and application production support teams. *Infrastructure Support* is focused on Kubernetes cluster and VM node monitoring and associated failure events will trigger workflow procedures to isolate the source of the event to find the root cause of reported failure. *Application Production Support* is monitoring failure events related to application container runtime deployments.

Instrumentation is configured to collect alerts from the various infrastructure layers, covering the VMware vSphere nodes, individual VMs (OS-level), Kubernetes platform, and associated containerized event monitoring services. A common approach for generating events at the instrumentation level is via configuration of threshold policies, based on specific resource counter value(s) or checks, which trigger an event notification based on some specified threshold event. In addition, native platform event service APIs can be leveraged to detect specific platform failure events. Events are aggregated by a centralized management system or cloud-based event management service, which supports further data processing such as event message enrichment, filtering, and deduplication. At this stage, event messages are being processed and root cause dependency analysis is performed to suppress events deemed. Event workflow processing often uses a reference datastore where information, such as support groups, pager numbers, and Slack channels, is maintained.

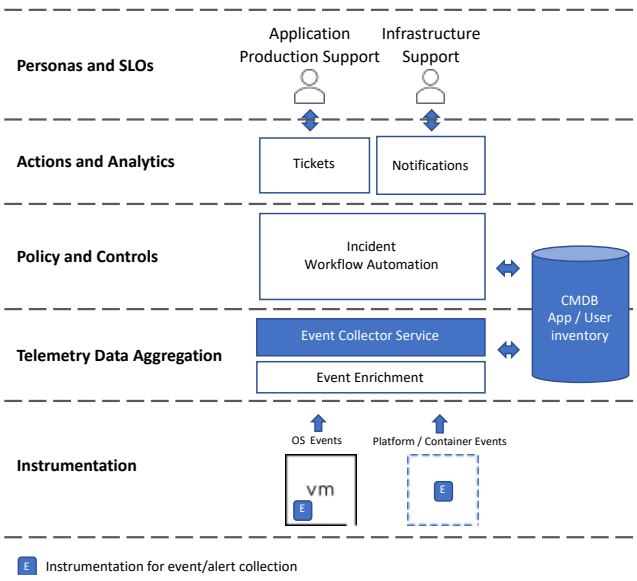

Figure 13 – Event and Alerting Data Pipeline Example.

In our example deployment collected failure events are being processed and, based on workflow automation rules, forwarded to either the *Infrastructure Support* or *Application Production Support* team members. For example, OS and Kubernetes platform failure events trigger creation of a ticket, which is assigned to the *Infrastructure Support* team. At the same time, email notifications are being sent to the *Application Production Support* team for awareness.

An important consideration for failure event management in cloud native environments is that single failure events only occur rarely. In real-world deployment environments multiple event (and log) messages are often generated during infrastructure failure conditions. One of the reasons for this phenomenon is that events are collected from different infrastructure technology layers and topology (parent-child) dependencies exist between these layers; failures in lower layers cause failures in higher layers. For example, assume a VMware vSphere server hosts multiple VMs and each VM hosts multiple containers. If there is a vSphere host failure, vSphere HA automatically restarts the hosted VMs on another vSphere host. During this restart period there can be service interruptions, and this is when the Kubernetes platform layer will also detect and alert on failure events.

Another consideration for event management in cloud native environments is that some failure events are related to transient failure conditions that are automatically resolved by (Kubernetes) platform and controller intelligence. In these scenarios, actionable failure events are scenarios where failures cannot be remediated automatically.

Current State of Observability

Software and vendor solutions for observability are rapidly evolving. Several vendors have rebranded their monitoring products as observability solutions. In addition, company acquisitions in the space of incident workflow management have enabled certain monitoring vendors to expand their product portfolio with workflow automation capabilities. In parallel, an increasing number of open source observability software solutions have become available.

Observability is an evolving technology trend that will most likely become more important due to the need for better monitoring visibility and control automation for distributed microservice applications and cloud infrastructure.

Here are some key observations about the current state of observability:

- **No universal definition (yet) for observability** – There are different viewpoints about observability. Although there is no single definition, there is consensus that it covers multiple telemetry data types, including metrics, logs, and traces, and that it provides a holistic system-level monitoring scope, covering both distributed applications and cloud infrastructure.
- **Multiple standardization efforts for observability** – Multiple CNCF projects are underway to standardize and open source functionality for collecting telemetry data. Some recent examples are the graduated projects from the Cloud Native Computing Foundation (CNCF) focused on container monitoring and application tracing; Prometheus, Jaeger, and Fluentd. In addition, the OpenTelemetry project is focused on developing APIs, libraries, agents, and collection services to capture distributed traces and metrics. The OpenMetrics project is focused on creating an open standard for collecting metrics. These efforts aim to prevent vendor lock-in and help enterprise IT organizations to develop the required level of instrumentation for implementing observability of their IT infrastructure and business applications. At the same time, the net result of the open source and standardization efforts is that agent instrumentation software is commoditizing and commercial vendors will most likely shift their focus on telemetry data ingest and value-added processing and data analytics, such as the ability to interrogate telemetry data using advanced analytical functions and query languages and customizable data visualizations.
- **Observability instrumentation is becoming day one requirement for new microservices** - The migration from application monoliths towards containerized microservices is still in the beginning stages for many enterprises. As part of the application modernization process, telemetry instrumentation for observability is becoming a day one requirement for identifying upstream and downstream application availability and performance issues. Observability code libraries and standard design patterns for structured logging, metrics, and tracing will facilitate and accelerate the adoption of application observability.
- **Move towards common backend (services) for different types of telemetry data** – Typical monitoring and observability deployments leverage multiple data stores for specific telemetry data types. For example, separate specialized data stores for storing logs and metrics. Especially SaaS-based monitoring backend services are starting to offer more flexibility in the types of telemetry data that can be ingested and support one common data store for different telemetry data types. Having access to a consolidated heterogenous set of telemetry data opens up the possibility for more advanced monitoring data analysis, such as cross-correlation among metrics, traces, and logs.
- **SaaS-based monitoring backend services gaining traction, but do require enterprise security** - With the growth of IT infrastructure and application deployments, the amount of telemetry data that is being collected is rapidly growing as well. Monitoring data store(s) must continuously expand to keep up with the data growth. Enterprise IT teams do not always have the skills and expertise for scaling and tuning of monitoring backend stores. As a result, some enterprises start to adopt SaaS-based monitoring services instead. To align with enterprise security and deployment guidelines, a key requirement for SaaS-based monitoring services is the support for security controls, such as role-based access control and data segmentation for multi-tenant user access.

- **SRE skillsets are still scarce and standalone ops function(s) still live on in many organizations** – Despite the industry momentum and mindshare, many enterprise IT organizations still rely on dedicated operations function(s), while trying to adopt agile automation and SRE skillsets. Transformation of IT organizations will take time and traditional IT organizational structures will likely continue to be in place for many enterprises in the foreseeable future.

Conclusions

Observability expands the scope of traditional IT monitoring by applying a system-level view to telemetry data and applying intelligence to this data to create actionable feedback loops.

Telemetry data intelligence can be used as input for controller and workflow automation, enabling automated changes and optimizations to cloud native infrastructure and applications. In addition, telemetry data intelligence can be used for reporting and search analytics by organizational stakeholders, also referred to as personas.

With IT organizations adopting cloud native deployments to support application deployment agility, observability is becoming a key solution component for bridging the gap between developers and operations support staff, providing similar toolsets to reduce friction and enable organization alignment.

The cloud observability framework presented in this paper can be used as a guide for evaluating existing monitoring architectures and developing new observability solutions, leveraging telemetry software instrumentation, monitoring tools, and data analytics.

In addition to cloud infrastructure and microservices, IT agility is also driving organizational change. Effective organizations require clear communication and delineation of responsibilities between infrastructure and application teams. Service Level Objectives (SLOs) clearly articulate boundary of concerns between organizational stakeholders, providing tangible and measurable service quality levels. Leveraging observability, telemetry data and analytics can be used for ongoing monitoring of SLOs, as common communication tool between organizations.

References

VMware Office of the CTO (OCTO) and BU publications:

- Application Enabled Infrastructure – Practical Cloud Native and the Rise of Application Platforms
- Introducing VMware NSX Service Mesh
- The Emergence of Cloud Runtimes: New Abstractions Driving Next Level of Cloud Reliability
- SLOs – The Emerging Universal Language in the Enterprise

VMware monitoring solutions for observability:

- Tanzu Observability (Wavefront) – Cloud-based metrics and tracing aggregation and analytics service.
- vRealize Log Insight Cloud – Cloud-based log aggregation and analytics service.
- Avi Networks (NSX ALB) - Web application analytics and transaction performance monitoring.

VMware Pivotal Spring integration with Tanzu observability:

- Sleuth - Spring Boot auto-configuration for distributed tracing.
- Wavefront - Spring Boot starter dependency module for sending metrics, traces/spans to Wavefront cloud service.

Observability open source instrumentation and tooling:

- Prometheus (metrics)
- Fluentd (logs)
- Jaeger (traces)
- Zipkin (traces)
- SkyWalkers (traces and metrics)

Observability standardization efforts:

- OpenTelemetry forum - Covers Opentracing and OpenSensus efforts.
- OpenMetrics project.

Acknowledgments

The author would like to thank the following reviewers for their valuable feedback:

Emad Benjamin, Senior Director and Chief Technologist, Application Platforms, Office of the CTO.

Andrew Tauber, Staff Technical Account Manager, TAM PSO field organization.

Lior Matkovitch, Director, Product Management, MAPBU.

Renate Kempf, Senior Staff Technical Writer, MAPBU.

Author: Harmen van der Linde, Sr. Director Cloud Architecture – Office of the CTO.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.

vmware®