Ericsson Technology Review

Charting the future of innovation

#4, April 2024

Cloud-native observability of telecom applications

Ericsson Technology Review — Charting the future of innovation | #4, April 2024

Cloud-native observability of telecom applications

Authors:
Jörg Aelken, Johan Wallin, Tadhg Deasy, Vincent Desbois, Magnus Standar

Solutions that provide observability for cloud-native telecom applications must be able to handle the dynamic, decoupled and highly distributed nature of telecommunication application components in the cloud-native environment. This requires new approaches to collecting, storing and processing telemetry data using and advancing the de facto standard technologies from the cloud-native open-source ecosystem.

ISSN 0014-0171 284 23-3409 | Uen

Ericsson, SE-164 83 Stockholm, Sweden
Phone: +46 10 719 0000

Application observability is a critical capability in network, system and service management, providing insights into an application's current state and supporting decision-making about changes or updates required to meet operational requirements.

The telemetry data exposed by an application is used as input to many management functions, such as assurance and optimization, allowing the operator to effectively maintain the network. Telemetry data is equally vital to developers, providing essential insights for runtime troubleshooting as well as in the transition to DevOps.

Since its inception, the telecom industry has always utilized a highly distributed architecture, even when capability was delivered in the form of fixed, integrated appliances. As telecom applications evolve from these fixed appliances to be deployed as cloud-native applications on independently provisioned platforms and infrastructure, it is crucial that observability methods and technologies also evolve to enable effective management of the applications, including their provided network functions. The desired flexibility and dynamic scaling of cloud-native applications mean that existing observability patterns will not be sufficient in terms of timely access to data nor in supporting the correlation of data across a disaggregated deployment.

In the face of this new, disaggregated deployment model for cloud-native applications, the generation, collection, persistence and post-processing of telemetry data must change to ensure that cloud-native applications are sufficiently observable. This includes the capability to correlate telemetry data from the applications and the underlying platforms and infrastructure. To ensure that data volumes are manageable, applications must expose appropriate volumes of telemetry data to support the wanted use cases, with the capability to contextualize the overall network deployed in an external higher layer management system.

Two sets of challenges to overcome

Cloud-native applications are designed so that their parts can be developed, deployed and scaled independently of each other [1]. These architecture and implementation choices are selected to enhance the speed of development and to achieve high system elasticity. However, these objectives are threatened by the high complexity of cloud-native applications and bring new challenges to the telecommunications industry. Any team that is responsible for ensuring the reliability of services that cloud-native applications provide must have the ability to handle:

1. Complex requests and operations that traverse a high number of deployed components
2. A multitude of moving parts due to independent life-cycle handling (some deployed components versions may coexist for the first time in production)

Terms and abbreviations

API – Application Programming Interface | **CaaS** – Container as a Service | **CNCF** – Cloud Native Computing Foundation | **CSP** – Communication Service Provider | **eBPF** – Extended Berkeley Packet Filter | **IaaS** – Infrastructure as a Service | **K8s** – Kubernetes | **OSS** – Operations Support Systems | **OTel** – OpenTelemetry | **OTLP** – OpenTelemetry Protocol | **PaaS** – Platform as a Service | **RED** – Rate, Errors, Duration | **SDK** – Software Development Kit | **USE** – Utilization, Saturation, Errors

3. Dynamic scale-out/in operations that change the dependencies of deployed components in runtime
4. Disruptive effects on the application caused by faults that originate from the highly layered and separately managed cloud infrastructure
5. An increase in reported problems that are unexpected and/or have unknown root causes
6. A huge amount of data to digest.

On top of these six generic technology-transformation challenges, communication service providers (CSPs) face three additional telecom-specific challenges. Traditional telecom applications are typically realized using well integrated nodes that provide standardized network functions, which expose telemetry data that describes the state of the functions provided. This observability follows traditional telecom management paradigms as specified in the ITU-T (International Telecommunication Union Telecommunication Standardization Sector) M.3100 Generic Network Information Model [2]. Therefore, the implementation of cloud-native telecom applications requires the ability to:

1. Collect enough telemetry data over costly transmission links from highly distributed systems
2. Efficiently monitor and troubleshoot an application in scenarios where trust and responsibilities may be shared between multiple business actors (the application owner, the cloud provider and Ericsson as the software vendor)
3. Continue to provide a functional view while simultaneously keeping the flexibility and detailed observability provided by the cloud-native approach.

The path toward cloud-native telecom application observability

According to the Cloud Native Computing Foundation (CNCF), "Observability is a system property that defines the degree to which the system can generate actionable insights. It allows users to understand a system's state from these external outputs and take (corrective) action" [3].

The CNCF definition has multiple implications. On the one hand, telemetry data must be collected from all parts of a system. On the other hand, telemetry data must be processed and provided to users, both humans and machines, to answer their many questions. The answers serve overall observability goals such as problem root-cause analysis, closed-loop enforcement, monitoring and so on.

The CNCF's definition also hints at different maturity levels in observability solutions. While it is important to detect when an application is under predictable conditions (the knowns), we also need a way to deal with what we do not understand and what we did not expect (the unknowns). This ambition level is connected to an organization's maturity and is equally relevant to its operational and software design business processes. **Figure 1** illustrates our model for observability maturity.

Telemetry data requirements

According to the cloud-native industry, the three pillars of observability are:

1. Metrics
2. Distributed traces
3. Logs.

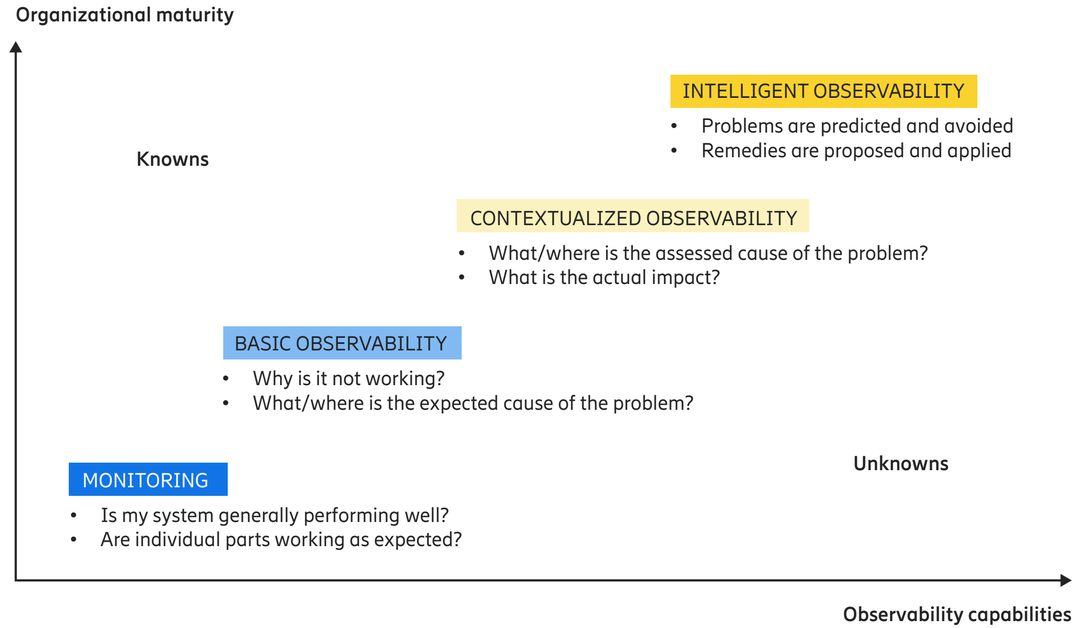

Figure 1: Observability maturity model

Combined telemetry data types based on these three pillars can significantly boost the maturity of an observability solution, provided this data is carefully designed and collected throughout the application, its platform and underlying infrastructure.

From Ericsson's perspective, there are two indispensable factors for success: the adoption of capable telemetry frameworks and the technical quality of the instrumented telemetry. The adoption of capable telemetry frameworks encompasses instrumentation libraries and telemetry collector components, providing key enablers such as:

- Data models that enable structured telemetry data
- Aligned time stamps (including synchronized time across highly distributed telecom networks)
- Aligned identifiers (resource identifiers, request/operation context identifiers)

	Telemetry data types			
Layer	**Metrics**	**Distributed traces**	**Logs**	**Other types**
Application functional layer	• Function-centric app custom metrics	n/a*	• Functional logs • Security and audit logs	• App business logic events
Application software layer	• Software-centric app custom metrics	• App distributed traces	• App container logs	• Continuous profiles
PaaS	• PaaS services custom metrics	• PaaS services distributed traces	• PaaS services logs	n/a
Caas	• K8s state metrics • K8s resource metrics • K8s control plane custom metrics	n/a	• K8s control plane logs • K8s audit logs	• K8s events • eBPF traces
IaaS	• Host resource metrics	n/a	• Host system logs	n/a

* Distributed traces for the application functional layer are captured in application-domain-specific and other types of business logic events.

Figure 2: Telemetry data types across the cloud-native horizontal layers

- Efficient streaming protocols, such as gRPC (Google Remote Procedure Call).

The technical quality of the instrumented telemetry can be assured through the use of complementary methods such as the four golden signals [4], RED (rate, errors, duration) [5] and USE (utilization, saturation, errors) [6], which set expectations on the scope of exposed metrics to achieve a decent observability level.

It is important to realize that the quest for high observability maturity does not end with the three pillars of metrics, distributed traces and logs. High observability maturity requires a multilayered perspective, as shown in the table in **Figure 2**. For example, to understand unexpected application performance problems, you must collect continuous profiles. To understand the root cause of Kubernetes (K8s) runtime failures or pod scheduling failures, you must collect K8s events deeper down in the stack.

The end of node-centric management

In the telecom domain, the integrated node has traditionally been the source of everything: the aggregated set of hardware, the singleton software package, the entry point for configuration, the resulting network function service and the emitter of the aggregated telemetry for all contained aspects. In the cloud-native paradigm, this changes: the life cycle of the network function service is separated from the hardware and software used to provide it – that is, what to achieve is separate from how to achieve it.

Cloudification separates shared infrastructure from shared platform services and dedicated (application) software. The separation of software from functional services has major consequences for both configuration and observability. Telemetry data is directly emitted independently from each disaggregated aspect of the system. There is no telecommunication node to provide aggregation. Instead, all telemetry data must carry its context as metadata to enable users to understand the system and get actionable insights.

To fulfill a service, infrastructure, platform and software resources must be orchestrated. In conformance with the cloud-native pet-and-cattle analogy, the continuation of the service (the what/the need) is what matters, while the hardware and software resources used to deliver it (the how) are disposable and interchangeable. As software instances take turns delivering the service, the software that is momentarily in use needs to emit telemetry both for the service it is currently delivering as well as for its own software observability. The platform and the infrastructure emit telemetry for their individual domains as well.

Because of the decoupling between the service and the software, telemetry for the service can be emitted by several different software instances over a period of time but still relate to the same service instance. Consequently, the service coupling needs to be provided by the emitted data itself; one cannot rely on the identity of any specific emitter to provide the service coupling. This behavior is not restricted to telecom – the same applies for software-as-a-service (SaaS) systems (or any aaS offer). The application instance execution is continuously served by software, but it should be possible to change the software instances that are used without noticeable impact on the application service delivery.

MONITORING METHODS

The objective of the four golden signals [4] is to monitor distributed systems with user-facing metrics:

- Latency (the time it takes to service a request)
- Traffic (a measure of how much demand is being placed on the system)
- Errors (the rate of requests that fail)
- Saturation (how "full" the service is).

The objective of the USE method [6] is to identify systemic bottlenecks quickly in early performance investigations based on the provision of the following metrics by any resource:

- Utilization (percentage of time that the resource was busy)
- Saturation (amount of work the resource has to do)
- Errors (count of error events).

The objective of the RED method [5] is to monitor microservices based on the provision of the following request-centric metrics:

- Rate (the number of requests per second)
- Errors (the number of failed requests per second)
- Duration (the amount of time these requests take).

Target architecture for observability data management

Figure 3 provides an illustration of our high-level functional target architecture for observability data management, which is designed to overcome all of the challenges and implement the necessary changes to support cloud-native solutions. It is comprised of four key functional blocks and the interfaces between them.

Observability data consumption

The observability data consumption block at the top of Figure 3 is where machines and humans consume the data that has been exposed by the observability data ingestion, processing and exposure block. Any data consumption that requires data from more than one data source must access the data from the observability data ingestion, processing and exposure block. Operations support systems (OSS) functionality such as assurance and automation functions, as well as other tools used to observe the behavior and state of the application, are typical for this block, along with business support systems functionality and other analytics or observability tools. External users such as monitoring or post-processing systems from CSPs also fall into this category.

Observability data ingestion, processing and exposure

The role of this functional block is to collect data from different sources according to the "collect data once, use many times" principle. It stores and distributes data using the appropriate data services and executes data processing that can be application specific (using rules provided by the applications), generic (configurable filtering or aggregation, for example) or both. This block also exposes data over relevant legacy or cloud-native management interfaces and provides control capabilities to configure data collection, as well as processing telemetry data from the realization layers and correlating/translating them into a functional view that is exposed through its northbound interfaces. Components of this functional block also fulfill the requirements of a data ingestion architecture [7].

The application

The application block is where business logic and supporting services run and execute their tasks, typically related to traffic handling, and where telemetry data generation takes place. Besides the actual business logic, there are typically telemetry agents that collect, filter and stream the data. Usually, there is also functionality that enriches the data records by adding metadata to enable traceability and to facilitate correlation on the receiving side. This block can be comprised of one-to-many workload component instances executing in one-to-many infrastructure clusters.

CaaS/infrastructure

The CaaS/infrastructure block serves as the execution environment for an application's workloads. It primarily consists of K8s clusters, but it can also contain infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) components as well as the hardware, depending on the deployment scenario. The multiple data sources of the platform and infrastructure telemetry data are located in this functional block and, as in the application, multiple telemetry agents collect and stream the data forward. This execution environment could be provided either in the form of data-center resources in a private cloud or as the corresponding services of a hyperscale cloud provider in a public cloud.

Interfaces

Several interfaces are needed to connect the functional blocks of the target architecture, namely:

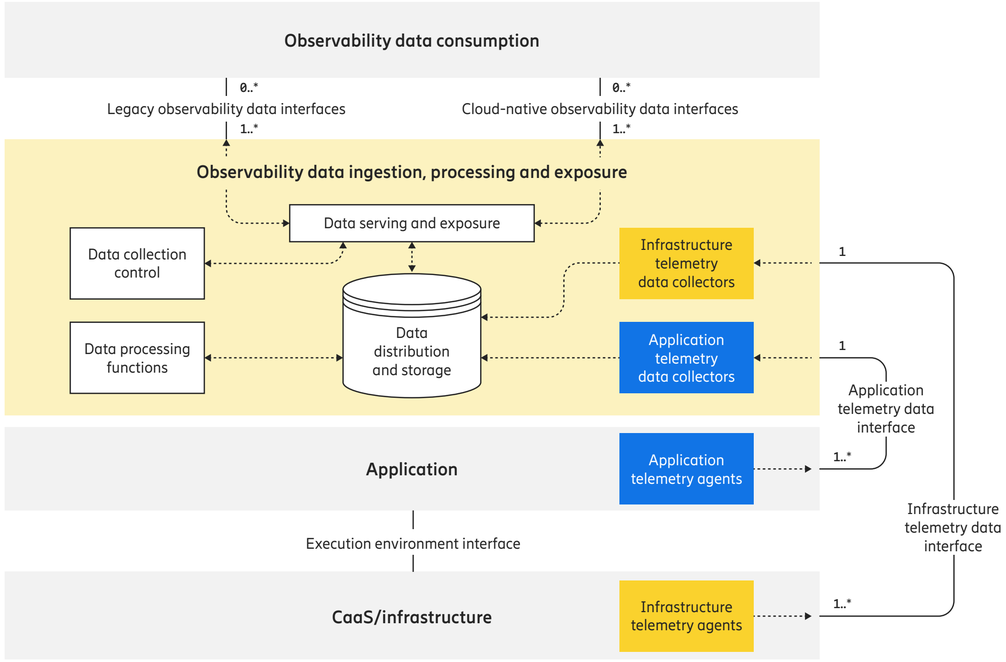

Figure 3: High-level functional target architecture

1. Legacy observability data interfaces
2. Cloud-native observability data interfaces
3. An application telemetry data interface
4. An infrastructure telemetry data interface
5. An execution environment interface.

Legacy observability data interfaces are needed to provide backward compatibility, which can be achieved with the help of different components that can expose services from different implementations compared with the existing solutions. These interfaces can be standardized or proprietary and expose, for example, traditional performance measurement data files or fault management alarms.

Cloud-native observability data interfaces are new interfaces that enable a data management paradigm that supports cloud-native deployments. These interfaces include those that expose the observability data plane and those that expose the observability control plane. They can be standardized or proprietary. Typical data plane interfaces in

this group expose telemetry log, metric and trace data, and support queries on them all. Control plane interfaces provide management capabilities for observability data schemas or configuration capabilities for the data collection.
The application telemetry data interface provides telemetry data (primary metrics/logs/traces/events) specific to the business logic.

The infrastructure telemetry data interface exposes telemetry data from the relevant infrastructure to correlate with the application to understand the functional behavior of the application business logic as well as to troubleshoot and determine the root cause for any deviation in functionality.

The execution environment interface is used by the application to retrieve the execution environment resource identifiers of the executing workloads. This is typically done through the K8s application programming interfaces (APIs). Note that these interfaces do not expose the infrastructure telemetry data, but rather the metadata needed to enrich the application telemetry data.

All of these interfaces require access control to ensure that only authorized users can access the specific data to which they are entitled. Furthermore, some of the interfaces may need to be visible only for certain user domains or tenants to enable a separation of public and non-public data exposure.

The OpenTelemetry project

The CNCF hosts a variety of open-source projects that are widely used in the industry and have made significant contributions to the development and standardization of cloud-native technologies, not least within the observability and analysis area. Ericsson has adopted several of these projects over the years.

The CNCF successfully incubated and graduated the Prometheus, Fluentd and Jaeger projects, achieving a high level of vendor neutrality from telemetry data instrumentation to telemetry data collection. However, those projects were essentially implemented as verticals with their own software development kits (SDKs), telemetry collectors and limited data alignment, which resulted in significant footprint and usability challenges. There were also open-source projects such as OpenCensus and OpenTracing, which had similar objectives but did not work well together, resulting in duplicated and wasted efforts.

The OTel project is relevant to the telecom domain for several reasons.

The OpenTelemetry (OTel) project was started with the objective of delivering a horizontally-aligned telemetry framework across the three observability pillars, while keeping the initial vendor-neutral objective. OpenCensus and OpenTracing were sunsetted and merged into OTel, which is now actively driven by the CNCF.

Figure 4 illustrates the scope of the OTel project, which has three key aspects:

1. It secures the availability of a unified telemetry instrumentation API and SDKs in all common programming languages.
2. It provides common data models and semantic conventions.
3. It provides efficient ways to collect, process and forward telemetry data thanks to the new OpenTelemetry Protocol (OTLP) and the very modular OTel Collector component.

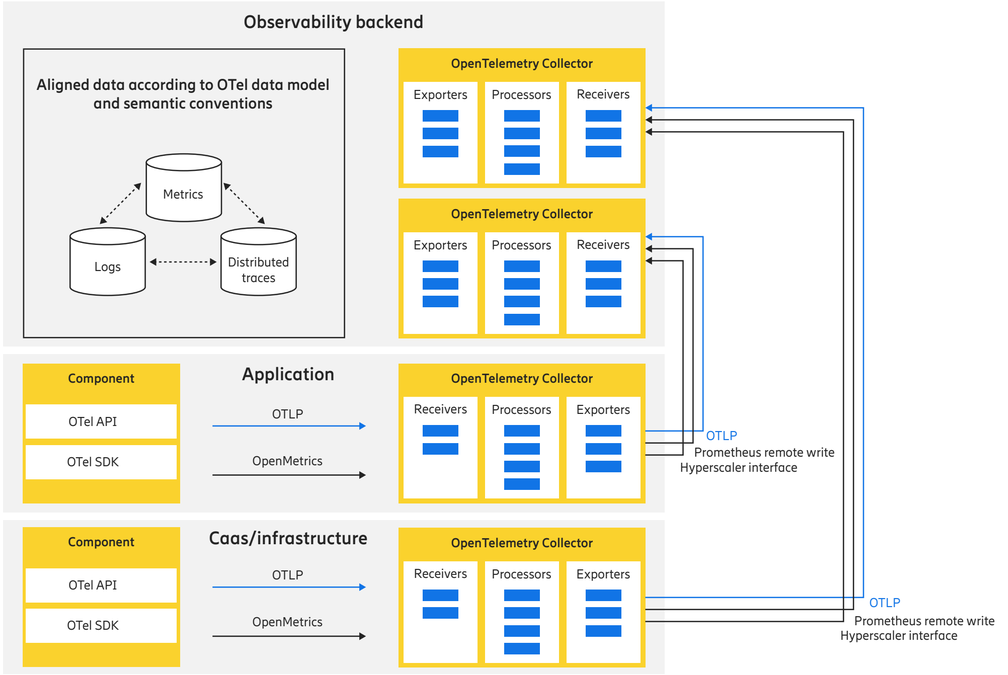

Figure 4: The scope of the OTel project

The OTel project is relevant to the telecom domain for several reasons. Firstly, the OTel Collector stateless agent deployment model, with a clear separation from stateful backend, will enable application footprint savings. Secondly, the ability to pick/choose/design suitable OTel Collector processors will help to reduce the emitted data volume. Thirdly, the end-to-end metric data push will result in latency reduction for metrics-based closed-loop automation. Lastly, data alignment will lead to a reduction in mean time to resolve.

Key benefits of cloud-native application observability

Cloud-native application observability can both improve existing use cases and enable new ones by:

1. Enhancing operational efficiency
2. Reducing integration effort
3. Enabling observability of distributed and disaggregated systems.

Enhancing operational efficiency

Cloud-native application observability enables significantly faster access to critical data. In contrast to legacy exposure mechanisms employed in telecoms, such as periodic file collection, cloud-native application observability operates at near-real-time speed. This reduction in latency of data collection in turn reduces the time to detect and diagnose suboptimal or undesirable network conditions, thereby facilitating quicker adjustments to the network.

Cloud-native application observability enables significantly faster access to critical data.

Faster data access is an essential prerequisite to effective automation, as it exposes a view of the network that is temporally proximate to the current state of the network. Use cases that can benefit include decisions around autonomous scale-in/scale-out, where quick decisions about the former can reduce the risk of wasted resources, while timely decisions about the latter can help to ensure that required capacity is available when needed. Decisions about power utilization and optimization can similarly benefit from timely access to data.

The opportunities presented in terms of operational benefits do not come without trade-offs and compromises. There is a capital expense in terms of compute, storage and networking resources to generate, transport and process telemetry data. Low-latency, streamed data may not be buffered/persisted for replay at source, which means that robust data management is essential to minimize the risk of data loss that would impede effective decision-making. Judicious usage of the telemetry control plane to manage data is required to deliver an acceptable balance between sufficient observability and manageable data volumes.

Reducing integration effort

The exposure mechanisms used by telecom applications at present comprise many diverse data encoding approaches. Observability data is exposed using a mix of encoding and transport mechanisms with SNMP (Simple Network Management Protocol) typically used for alarm/alert-style data, file or TCP (Transmission Control Protocol) streams used for metrics and a disparate set of (non) standards for log and trace data. Telecom equipment vendors and CSPs incur recurring costs to develop, support and operate networks that expose data in this fashion.

Cloud-native communities, such as the CNCF, actively encourage the use of standardized approaches to observability data management. Common aspects of metrics, logs and distributed traces are handled in a common fashion; the schemas used to encode/decode the data are common, as is the transport mechanism. This reduces the effort to embed the capability for telecom equipment vendors within their offerings and enables easier integration into tool chains that are designed to utilize such data. OpenTelemetry is fast becoming the de facto standard in this space.

Enabling observability of distributed and disaggregated systems

As noted, cloud-native applications may be deployed in a distributed manner within a certain context (such as a namespace). The use of OTel APIs, SDKs and tools, which inculcate the use of effective distributed tracing, eases the correlation of data across applications deployed using a distributed architecture.

Similarly, the use of cloud-native application observability facilitates the correlation of application telemetry with telemetry sourced from the typically independently provisioned and operated runtime infrastructure on which the applications are deployed. Frameworks such as OTel and cloud-native observability techniques are being adopted by cloud-native infrastructure providers, including hyperscalers. Embracing an aligned approach across the application and infrastructure layers reduces the challenge of managing and correlating data across application and infrastructure, thereby reducing R&D spending.

Conclusion

Ericsson agrees with the cloud-native community view that the purpose of application observability is to provide actionable insights into an application's performance and characteristics. These insights, gained by correlating data from different, independent and self-contained data sources, can be used in assurance loops that optimize the application's behavior, do trend analysis to predict future needs and, in case of some anomaly, troubleshoot the application. We see this approach as crucial to achieve our goal of a highly automated network managed by closed-loop telecom applications driven by artificial intelligence.

We acknowledge the disaggregated nature of the telecom applications and the consequences with respect to the continuity of the application being captured in its data as opposed to the continuity of its individual parts. Therefore, the collection, persistence, processing and exposure of the telemetry data needs to be changed, utilizing cloud-native technologies such as those available as de facto standards from the open-source ecosystem. Aligned data exposure better supports correlation, post-processing and consumption of the data originating from multiple, disaggregated sources, such as applications, platforms and infrastructure.

Ericsson has a strong history of providing effective observability for our applications. To protect and grow that capability into the future, we must evolve our observability solutions together with our customers, by embracing these new and exciting cloud-native paradigms and technologies.

The authors

Jörg Aelken is an expert in cloud-native orchestration and life-cycle management whose work focuses on architectures and solutions for the management of telecom applications utilizing cloud-native technologies. He joined Ericsson in 1992 and is a member of the BCSS Architecture and Technology organization. He has held several positions in R&D, most recently related to virtualization and cloud, including serving as standardization delegate. Aelken holds an M.Sc. (Dipl.-Ing.) in electrical engineering and communication technology from RWTH Aachen University in Germany.

Johan Wallin is an expert in network management at Group Function Technology, Architecture. He joined Ericsson in 1989 and has held several various positions in R&D, working with network management in several aspects as well as general network architecture. Wallin holds an M.Sc. in computer science and engineering from the Institute of Technology at Linköping University in Sweden.

Tadhg Deasy is an architect in Business Area Cloud Software and Services. He joined Ericsson in 2006 and has worked primarily within OSS and network management. In his current role, he focuses on network assurance and the role of management systems in assurance. Deasy holds a B.E. in electronic engineering from University College Dublin in Ireland and a Ph.D. in engineering from Queen's University Belfast, the UK.

Vincent Desbois is a cloud-native observability senior specialist in Business Area Cloud Software and Services. He joined Ericsson in 2000 and has worked in R&D with network management, process and information architecture, as well as usability matters in open-platforms-based applications. In his current role, he focuses on observability for cloud-native applications. Desbois holds a M.Sc. (Dipl.-Ing.) in electronic, electric and computing engineering from the ESIGELEC School of Engineering in Rouen, France.

Magnus Standar is an architect for Ericsson Cloud RAN (radio access network). He joined Ericsson in 1998 as a developer of Mobitex and has since held various software and systems architect roles for Wideband Code Division Multiple Access, Long Term Evolution and New Radio (NR) RANs. In his current role, he focuses on the architectural impacts of cloudification of NR RANs. Standar holds a university certification in electronic engineering from Chalmers University of Technology in Gothenburg, Sweden, and a B.Sc. in computer science from Linnaeus University in Växjö, Sweden.

References

1. Ericsson Technology Review, 5G architecture for hybrid and multi-cloud environments, 29 March 2022, Alonso, A.; Saavedra Persson, H.; Kassaei, H ↗
2. ITU-T, M.3100 (04/2005), Generic network information model ↗
3. CNCF Observability definition ↗
4. Google, SRE book, The four golden signals ↗
5. London Microservices User Group (02/12/2015), Monitoring Microservices, Wilkie, T ↗
6. The USE Method, Gregg, B ↗
7. Ericsson Technology Review, Data ingestion architecture for telecom applications, 16 March 2021, Rönnberg, A-K; Åström, B; Gecer, B ↗

Further reading

- Ericsson, Cloud-native applications ↗
- Ericsson blog, Cloud-native applications framework for 5G ↗
- Ericsson Technology Review, Cloud-native application design in the telecom domain ↗