October 2025

A Cloud Native Telco Forum Initiative

CNTF Productivity Gains

Revision 2.1.0

Responsible Officer / Initiator
Lukas Leuthold, Swisscom

Contributors
Deutsche Telekom
Daniel Dierich, Kai Steuernagel, Mohamed Annis Souames
Telefònica
Matthias Sauders, Bas Hendrikx
Swisscom
Lukas Leuthold, Adrian Kurt, Josua Hiller, Joel Studler, Ashan Senevirathne
Ericsson
Peter Wörndle, Jonas Falkena, Staffan Bonnier, Anders Fagerlind, Jawwad Rasheed

CNTF participating Operators
Deutsche Telekom, Vodafone DE, KPN, TNO, Telenor, Vodafone IT/Fastweb, Orange FR, Container Solutions GB

CNTF participating vendors
Ericsson, Nokia

Content

General Information

Purpose

This document provides a comprehensive exploration of the significant productivity gains resulting from the adoption of cloud-native principles. It specifically highlights the relevance of these gains in the context of 2025. By synthesizing the individual experiences of CNTF associates, a unified narrative is presented, offering a holistic view of the transformative potential.
In embracing cloud-native principles, it is crucial to recognize that this journey extends beyond mere technical considerations into ways of working and processes. While technical aspects are vital, the adoption of cloud-native principles by technology suppliers and CSPs should also be driven by the objective of generating tangible business value. This broader perspective encourages organizations to explore the full spectrum of potential benefits associated with cloud-native transformations.
The productivity gains outlined in this document represent a network of interconnected benefits, reinforcing and complementing one another. Each gain has the potential to facilitate the achievement of other gains, further amplifying the positive impact. By strategically leveraging cloud-native principles, organizations can anticipate substantial productivity improvements in 2025, ensuring they stay at the forefront of innovation and efficiency in a rapidly evolving digital landscape.
CNTF drives solutions and alignment that scales for both vendors and operators. Creating a healthy industry with room for differentiation where it makes sense, while aligning for reduced cost in other areas. Also includes high level alignment of industry pacing/timing where needed.

Scope

The content applies to the CNTF initiative, started in November 2024 by Swisscom. Being a forum, we aligned the content among all CSP and ISV listed on the title page. The findings shall serve as a guideline and be actively discussed with our ISV, CSPs and Cloud Providers. While the guidelines are applicable for both network functions/platform and software engineering in general, examples and pain-points are coming from network side.

Goal

The primary goal of this document is to express the common pain-points encountered by the CNTF associates and share and align the expected user experience within the cloud-native context. We realise that CSPs need to amplify direction/goals for WoW and process alignment. The underlying vision of this document remains solution- and product-agnostic, aiming to harmonize a unified experience across different operators. By presenting these pain-points, we intend to provide a focused and condensed articulation of the areas where software suppliers (both vendors and F/OSS) should allocate resources and make improvements. At the same time, we present essential characteristics of cloud-native operations (ways-of-working and processes), applicable to CSPs wanting to make most use out of cloud-native technologies.
The topic of cloud nativeness and its design patterns has a high potential of becoming a trite cliché if not related to its potential to gain the operational productivity significantly. There is also a risk that an uncoordinated introduction of cloud native principles may lead to a high level of industry fragmentation. To counter this perception and these risks, we firmly assert that embracing cloud-native principles has the potential to stimulate significant productivity growth. As such, our objective is to go beyond general statements and explicitly specify the gains in productivity that can be achieved, quantify them where possible, highlight existing challenges, and indicate areas of alignment within the CNTF perspective.
Given the extensive range of aspects and disciplines within cloud-native principles, attempting to cover them all in a single document would exceed its intended purpose. Therefore, for the first edition of this document, we have chosen to focus on three key areas that we consider of utmost importance. By prioritizing these areas, we aim to foster continued dialogue, both within the CNTF community and with vendors, to further enhance our understanding and collaboration in advancing cloud-native practices.

Document History

SemVer	Date	Description
1.0.0	CW11/25	Initial release
2.0.0	CW15/25	Chapters / Area restructured. Always-on-latest-SW Paradigm, “No-touch”/Continuous Flow SW and Feedback Loop, Testing in Production, CNTF proposed foundation services for GitOps, Multi-App-On-Single Cluster, DORA Metrics for CD.
2.1.0	CW44/25	New Areas: Observability – Area 3, Scalability – Area 4, High Availability Area 5

Glossary

ISV	Industry Software Vendor, also NF vendors, Supplier
CSP	Communication Service Provider, also Telcos, Operators
F/OSS	Free- and Open Source Software
CNTF	Cloud Native Telco Forum
CNCF	Cloud Native Computing Foundation

Definition of Terms

Software/Realization lifecycle management	The process of changing software artifacts or software related configuration e.g. security update or Pod replica count.
Functional lifecycle management	The process of changing the functionality a network function provides in the network e.g. creating a new APN/DNN or enabling a new 3GPP interface.

Area 1 - Software Freshness

1 - 1 Definition

Software freshness refers to the extent to which software is kept up-to-date, secure, and in line with current standards, technologies, and requirements. It is essential to maintain optimal performance, compatibility, and safety.
To illustrate this, we can use the analogy of the cold chain, which is a temperature-controlled supply chain for perishable goods, ensuring their effectiveness and safety. Similarly, software freshness ensures that the perishable aspects of software, such as security, performance, and relevance, are preserved through continuous maintenance and updates.
It is crucial to regularly update software with patches, updates, and dependency versions to avoid any degradation in performance or (regulatory pushed) security. Neglecting these updates can pose significant risks, including security breaches, bugs, or becoming obsolete.
By adopting a continuous improvement approach with smaller and incremental changes across development, delivery, deployment and operations, it becomes easier to identify and fix issues in the early stages of development. This approach ensures that the software remains stable and of high quality over time. When updates are small and frequent, any introduced bugs are more likely to affect only a limited part of the system, minimizing the potential impact on the overall system. Consequently, it becomes easier to isolate and resolve issues without causing widespread disruption.
Software Freshness encompasses several key elements:

1. **Always on Latest Software**

   The goal to be always on Latest Software and avoid touching software and/or configuration by hand. This applies to CSP taking that latest software from ISV, as well as for ISV to provide the latest stable version of supplied open-source software components.

2. **Separation of Software Life-cycle Management and Functional Life-cycle Management**

   The ability to manage the software realization of a network function independent of the functionality it provides to the network. This means that a software package can be upgraded without impacting the function of the CNF in the network. Vice versa, it should be possible to turn on a functionality available in the software without changing the realization in software.

3. **Continuous Software Flow**

   Developing small and lightweight software artifacts with a simple structure that enables fast and stable CI/CD pipelines. By incorporating widely used Cloud Native standards and best practices, such as OCI and Semantic Versioning, it ensures smooth operations and reduces operational expenses. Vendors should also support the integration of selected lifecycle automation tools to streamline processes.

4. **In-service Software Update (ISSU)**

   Effective and seamless software lifecycle management is crucial for maintaining software freshness, security, and performance. Supporting in-service software and configuration upgrades allows for seamless updates without disrupting ongoing sessions. Changes should be as small and incremental as possible to help keep ISSU simpler and more robust. To this end, the recommendation for CSPs who are not yet ready to always run on the Latest Software, to resort to other upgrade mechanisms than ISSU, e.g. utilizing network level redundancy to maintain services during the upgrade procedure.

5. **In-service Software Roll Forward / Rollback (ISSR)**

   As part of the software flow, the ability to seamlessly revert changes, whilst maintaining the NFs service is essential. The underlying strategies may vary including a roll forward to a fixed version or a roll back to apply a known-to-work software version or configuration. As for ISSU, the smaller the changes are the more robust this procedure is.

6. **In-service Software Downgrade (ISSD)**

   In certain operational scenarios, the ability to downgrade software in-service is required to ensure continued network availability and adherence to service level expectations. An ISSD approach enables CSPs to revert to a previous software baseline while maintaining active sessions and without forcing a full service disruption. Similar to in-service upgrades and rollbacks, downgrade operations are most effective when changes between releases are kept minimal, reducing incompatibility risks and operational complexity. ISSD procedures may be triggered due to functional regressions, performance degradation, security vulnerabilities, or unforeseen behavioural changes introduced in newer releases. To ensure robustness, CSPs should verify backward compatibility across data models, configuration schemas, and state persistence. Where this cannot be guaranteed, network-level redundancy mechanisms may be leveraged to safeguard service continuity during the downgrade process.

7. **Release Strategies**

   With the microservice architecture, a canary upgrade can be implemented to test new features/versionson a limited set of subscribers. This accelerates the deployment of changes into production in a more stable manner with a smaller blast radius. Testing will then happen either in staging/canary/labs or production. This means to accept failures in the release and having robust recovery mechanisms (as in A/B, canary…). Canary can happen on node (NF) level or CNF internal. We expect to align which strategy is being used in the industry to avoid double implementations and to understand the upsides and downsides of each option. There is the potential to roll-update whole NF sets.
   Another key point is the separate “Functional” and “Realisation” levels.

   - **Functional** means to enable and expose a new feature/function to a limited or selected set of users (e.g. for 5G Core Network Platform: by UE types or IMSI range or Network Slice NSSAI).
   - **Realisation** means to bring in new software to the system only doing regression verification.

8. **Continuous Feedback Flow to ISV or F/OSS community**

   Cloud Native brings increased focus on fast delivery and deployment of updated software components. There is an expectation of fast turn-around times and to facilitate that it is of highest importance to have an established feedback process to close to loop. An automatic continuous feedback flow from CSP to ISV is essential to achieve high quality software and fast turn-around when errors occur or security vulnerabilities need to be remedied.

   Different types of data need to be collected and made available to ISV for different purposes:
   - Software Pipeline Status and Health
   - Artefact Pipeline Progress
   - Cloud Infrastructure Configuration
   - Software Configuration
   - Application Configuration
   - Application Feature Usage Information
   - Test Result Information
   - Traffic and Load Information
   - Metrics and KPIs for Trend Detection
   - Troubleshooting Information (e.g. logs, traces)

   Source of feedback information can be CSP lab, staging and production environments. This enables the possibility to establish a lean pre-production automatic early feedback channel back to the Independent Software Vendor.
   Agreements between information source owners and information receivers must be established. To adhere to regulatory requirements and privacy principles some information may have to be anonymized before transferring from CSP to ISV.

9. **Division of Responsibility – Optimal Verification and Integration**

   Quality is assured as joint effort between ISV and CSP. Automated quality assurance activities will be conducted both in ISV and CSP environments, selecting the most appropriate environment for the task at hand. Generally applicable verification shall be shifted towards ISV, while CSP specific verification shall be

shifted towards CSP. An aligned staging process and an established feedback loop is essential for this to be efficient.
The ISV defines the versions of opensource projects verified and supported in a product to ensure coherent integration and quality assurance. The cloud-native SW architecture can allow for replacement of opensource components by upstream versions, which in turn requires a modified responsibility split (e.g. SLA) between supplier and operator for the product, in which case E2E responsibilities will shift from the supplier to the operator.

10. **Aligned Staging Process**

    To streamline the software flow from supplier to operations, staging environments are an essential asset. Staging environments exist both on the supplier side for testing and integration as well as on the operations side for integration into the local operations environment. We aim for an alignment of essential steps in the staging process to ensure consistent quality assurance and verification in an end-to-end process whilst avoiding duplicate activities.

11. **Aligned Test Automation**

    An aligned test scope between supplier and operator is critical for an efficient feedback flow in which e.g. reference tests can be used to detect anomalies. Implementing automated testing after each software drop is essential. It should be integrated into the continuous integration/continuous deployment pipelines, providing a fully autonomous testing experience.

12. **Regulatory Push for Security 30/60/90**

    This typically implies a staged plan or timeline to improve its cybersecurity posture. This might issue a 30/60/90-day remediation plan:
    By Day 30: Patch all high CVSS-score vulnerabilities.
    By Day 60: Implement MFA and logging for critical systems.
    By Day 90: Conduct a full risk assessment and train all staff.

13. **Built for Automation**

    An efficient low-friction software flow requires all components and processes along the way to be built for automation. The software is provided with APIs for configuration alongside machine readable assets. The processes for ingesting, testing and deploying software in production defined by the operator do not include any manual steps and are fully automated and integrated with the automation provided by the Network Functions.

14. **Non-Technical aspects**

    A lot of time is spent for assessing network impacts for a change, approvals in the process, scheduling maintenance activities, project planning and documentation. This will also define the delivery and ingestion process as such and how to first align, then streamline, and finally automate the process.

15. **Bringing ISV Dev to CSP DevOps in a scalable manner**

    There are many examples of the successful use of cloud native principles and technologies in the IT industry, when Dev and Ops cooperate and optimize their activities for a single deployment environment accessible by both parties. This is different from a Telco setting with an N:M relation between ISVs and CSPs and intermediate delivery steps of complex software. This calls for a high level of alignment in terms of technology and WoW across the N and M, else the introduction of cloud native principles may not scale well and turn out counterproductive.

16. **Accessibility of CSP stage (labs/tenants/slices) from ISV Dev side**

    To facilitate efficient troubleshooting and to streamline the feedback loop, the ability for an ISV developer to gather information on a CSP deployment is crucial. In addition to providing information about a deployment (see item 7) the ability to securely access CSP staging environments from ISV side in a controlled manner to gather more detailed data is needed for certain cases.

By embracing these practices, the software can maintain its freshness, ensuring it remains up-to-date, secure, and aligned with current standards and requirements.

1 - 2 External References

- https://github.com/lfn-cnti/bestpractices/blob/main/doc/whitepaper/Accelerating_Cloud_Native_in_Telco.md
- https://www.ngmn.org/highlight/ngmn-publishes-cloud-native-manifesto.html

1 - 3 Current Pain Points (What's difficult)

We feel very limited to embrace Software Freshness because the delivery is still designed in a way that people "touch it". The reality is the many traditional support and commissioning methods of the ISV are still needed. Software drops are too heavy, too long (low cadence) due to low CSP adoption of new software and industry requirements on robustness on CSP and vendor alike. The above definition is not realizable and CSPs must in-house develop tools and processes that would be best placed at vendor side for reusability and scale.

1 - 4 Harmonized Productivity Gains Quantified

Swisscom has demonstrated a remarkable reduction in the time required for manual deployment, reducing the process from 2.5 weeks to just 10 minutes. The significant progress made in this area highlights the potential to save approximately 8,000 hours of DevOps work within the DMC context in 2025. This accomplishment reinforces the existing pain points and emphasizes the scope for further productivity improvements. Our focus on addressing these challenges empowers us to enhance our overall efficiency and cultivate even greater productivity gains.

1 - 5 CNTF's ask to ISV or F/OSS

AREQ A1-1.001	As expressed later in Area 2 the transition from an imperative to a declarative intent driven methodology where the desired state of the system is defined in a simple and abstract intent in Git repositories. Simplify LLDs to expose to engineers/service-mgmt tools, abstracted intent to reduce manual interventions and complex dependencies. Artifacts should be machine readable (MOPs or NIR)
AREQ A1-1.002	Software images should be delivered via OCI
AREQ A1-1.003	Software releases should follow Semantic Versioning
AREQ A1-1.004	Software should be delivered with testing tools that allows easy automation of integration testing in the Operator's environment
AREQ A1-1.005	Canary rollouts should be supported by the Software
AREQ A1-1.006	Vendor to show the granularity and timing of SW deliverables
AREQ A1-1.xxx	Availability of machine-readable delivery assets e.g. release notes
AREQ A1-1.xxx	Strive to deliver the latest stable version of supplied open-source software.
AREQ A1-1.xxx	Define relevant KPIs and data sets that are suitable to receive as feedback from CSP.
AREQ A1-1.xxx	Provide means to anonymize sensitive or personal information collected from the system.
AREQ A1-1.xxx	Develop tools and processes to handle information received from CSP in a safe manner.

1 - 6 CNTF's ask to CSP readiness

AREQ A1-2.001	Ability to run automatic regression testing
AREQ A1-2.002	Pipeline capabilities
AREQ A1-2.003	Availability of "testing" nodes: lab or canary
AREQ A1-2.004	Availability of staging process and zone
AREQ A1-2.005	Fully automated SW ingestion process / Software ingestion and update flow without manual steps defined
AREQ A1-2.006	Approval flow via Git
AREQ A1-2.xxx	Ability to automatically process and act on machine readable delivery assets e.g. release notes
AREQ A1-2.xxx	Willingness to share relevant feedback information.
AREQ A1-2.xxx	Strive to deploy the latest stable version of ISV supplied software.
AREQ A1-2.xxx	Automatically collect feedback data and make it available to ISV.
AREQ A1-2.xxx	Provide necessary correlation information for ISV to be able to interpret feedback data.

1 - 7 Testimonies

Swisscom has undergone a remarkable transformation, transitioning from productized CI/CD pipelines to a GitOps-centric approach. This shift has resulted in substantial improvements, with manual deployment time reduced from 2.5 weeks to a mere 10 minutes. However, the telco software industry still heavily relies on traditional manual practices. In the current process, software is delivered to a specific endpoint, requiring an engineer to manually copy it to an internal software repository. Furthermore, highly specialized ISV engineers must prepare and modify the Day0 and DayX configurations to facilitate software deployment to a DEV staging instance.

One significant challenge is that the software delivery process is still designed in a way that an expert must "**touch it**," representing a bottleneck in the process. This reliance on expertise hinders the efficiency of software delivery and does not align with our Go-To-Market targets due to its time-consuming nature.

In embracing the concept of Software Freshness, Swisscom acknowledges that fully tested software from the ISV might not be expected. Instead, we aim to enable basic local testing, even at the level of an engineer's laptop. Our Continuous Testing/Verification tools automate the testing of each software drop. By adopting this approach, the ISV can deliver preliminary software versions to be tested in a production-like environment, expediting the overall software delivery process.

Lukas Leuthold, Swisscom

Area 2 – Declarative Management

2 - 1 Definition

Configuration management for cloud-native systems across multiple layers involves processes, tools, and best practices for managing deployment, configuration and ongoing operations.
The layers are typically distinguished as follows, but may slightly vary in a specific deployment architecture

1. **Functional/application layer**
   configuration of the actual functionality a network function provides to the network
2. **Application/CNF software layer**
   configuration of the software composition and non-functional attributes of the software
3. **Infrastructure layer**
   configuration of the infrastructure hosting the application/CNF including low-level connectivity and compute resources

The application/CNF software layer and infrastructure layer comprise the realization of a specific network function. Each layer, functional, application software and infrastructure, may cycle through different stages of configuration independently i.e. all layers have their own Day 0, 1, 2/n configurations.
In dynamic, distributed environments, there are many parameters that need to be planned, defined and provisioned and managed across versions, generations, change-management integration, validation and reconciled upon the live state to desired state discrepancies.
Note: User data management (HSS/UDM subscriber data provisioning) is using dedicated provisioning interfaces and is out of scope of this analysis
A cloud-native approach to configuration management would be:

- Moving away from direct imperative and sequential configuration in a day 0, 1, 2/n manner based on e.g. NETCONF, CLI, SNMP to a declarative source-of-truth mastered in GitOps and synced and reconciled e.g. with K8s operators.
- Non-repetitive configuration (day 0/1 templating) should be driven using Kubernetes ConfigMaps or Custom Resources with Schema.
- A network function should expose declarative APIs for configuration.
- Kubernetes Operators with specific scopes, e.g. provided by the NF supplier, should be used as an abstraction front-end for configuration. ETSI NFV defined functions such as VNFM/VNFO/EMS is put into many operators and such traditional dedicated systems will disappear.
- Management systems can access the networks via the abstraction layer based on K8s CRD/Operators.
- Instead of managing a flat model of well over 1000 parameters per Node, abstract configuration artifacts e.g. CRDs should be provided that reflect repetitive use cases of the business. Examples from the 5G Core Network Platform of such repetitive configuration tasks are:
  1. APN as a service (E2E incl. HSS/UDM/DNS and Packet Gateway)
  2. Packet inspection and service chaining/logic rules exposed as a service
  3. Service-aware charging rules exposed as a service
  4. Network Slicing (NSSAI/URSP) as a service (E2E)
  5. Radio Resource Partitioning as a service (on MME/AMF)
  6. Routing behaviour for P-GW/UPF contexts used by Private Networks.
  7. Installation of a new data plan (PCF)
  8. Management of Secrets & Certificates (E2E)
  9. Management of telemetry profiles (destination, datapoints, periodicity etc)
  10. IP/FQDN Claims for day 0 instantiation
  11. Primary & Secondary Network configuration incl. L2 provisioning
  12. Deployment of VMs or K8s Cluster as a Service and Namespace as a Service

2 - 2 External References

Multi-Intent Service Configuration Vision Whitepaper by L. Leuthold – July 22

2 - 3 Pain Points (What's difficult)

Imperative protocols (Netconf, SNMP, CLI) instead of declarative cloud-native approaches (configmaps or Custom Resource Descriptions) lead to fire-and-forget operations and higher complexity (e.g. through configuration sequences) in configuration management. There are no APIs provided by the network functions to plan and design service logics and data plans characteristics.
Even having cloud native configuration management, it is still difficult to have an E2E service configuration from a design studio point of view. We believe that creating sections of configuration each having their own operator exposing APIs will help.

Imperative Methodology
The current infrastructure deployment processes for K8s clusters and Switch Fabric configuration in Bare-Metal solutions are driven by an imperative, fire-and-forget methodology. This approach involves one-time push operations without continuous pull-based reconciliation. The lack of ongoing synchronization between the actual system state and the desired state defined in Git repositories can result in potential drifts and the need for manual corrections.

Manual Interventions
Despite the integration of automation pipelines, significant manual interventions are still required in the overall process. Manually creating complex Low-Level Designs (LLDs) and pre-defining all the necessary information can be time-consuming and prone to errors. The current systems do not facilitate seamless and efficient changes, as configurations are often spread throughout intricate LLDs and input information is spread throughout the organization, making updates burdensome and slow.

End-to-End Automation Capability
Many deployments lack comprehensive end-to-end automation capabilities and coherent, streamlined processes, from Kubernetes cluster deployment to lifecycle management (LCM). This limitation translates into manual and error-prone efforts and waiting times due to task handovers and scheduling dependencies. Tasks such as cluster management, bootstrapping sidecars, overhead of management nodes, application-directed networking requirements (e.g., L2 isolation, LAG, Active-Passive NICs, SR-IOV vs. OV), and provisioning additional VLANs on L2 fabric in a cloud-native configuration management approach (as detailed in Area 2) add to the complexity and challenges faced.

Vendor Tooling Landscape
The reliance on forked Open-Source tools within the tooling landscape provided by vendors has two consequences. Firstly, vendors must maintain their own forks and port upstream features into their customized versions, resulting in additional efforts and potential discrepancies. Secondly, telcos face difficulties in leveraging publicly available knowledge of the tools since the vendor-specific forks limit access to widely shared expertise and documentation. Addressing these pain-points is crucial to streamline deployment processes, reduce manual interventions, and embrace end-to-end automation that aligns with cloud-native principles. By reevaluating the methodology, enhancing tooling landscapes, and focusing on standardization and continuous reconciliation, Swisscom can overcome these challenges and improve operational efficiency.

2 - 4 Harmonized Productivity Gains Quantified

Not possible as of now. This transformation has just started – we do not have success testimonies other than showcasing SDC and IaaS automation

2 - 5 CNTF's ask to ISV or F/OSS

AREQ A2-1.001	IaaS Provisioning should be exposed by API or CRD/KRM based (Cluster API) and not performed by proprietary (somewhat closed) tools. This includes E2E Networking from MetalLB, external Networks (L2 fabrics) and remote order entry to other domains (such as DC networking)
AREQ A2-1.002	Implement Kubernetes Operators to automate the management of infrastructure components, reducing the need for manual pre-definitions.
AREQ A2-1.003	Continuously reconcile the actual system state with the desired state defined in Git repositories
AREQ A2-1.004	Provide cloud native APIs to hydrate and push desired state to running state for supported infrastructure components including testing
AREQ A2-1.005	Provide software enabling CSPs to manage the lifecycle of the clusters, handle updates, and ensure that the desired state is maintained

2 - 6 CNTF's ask to CSP readiness

AREQ A2-2.001	Git-based process for configuration management i.e. all configuration is applied via Git and NF-local configuration procedure (CLI) are disabled by default and only used in emergencies
AREQ A2-2.002	Availability of observability systems to secure correct operations of the network functions beyond monitoring of the desired state
AREQ A2-2.003	Input data for LLDs is available via APIs or stored in Git
AREQ A2-2.004	Harmonization of tooling landscape

2 - 7 Testimonies

Swisscom has used SDC (schema driven configuration) to build a bridge from NETCONF to KRM based approach. With that, we can keep the configuration in GitOps and invest in the config management on service orchestration level to carry and provide management facility that are derived from the catalog, inventory or by user administrations. Config generation is done using Jinja2 Templates which allowed Swisscom to improve quality of day0 and day1 configurations and better align across environments (development, staging, production).

Swisscom is actively developing automation processes to address limitations in the current vendor-provided solution. The existing approach requires complex static Low-Level Designs (LLDs) upfront for deploying and upgrading Kubernetes (k8s) clusters. However, the initial deployment cannot be seamlessly integrated with the necessary post-deployment steps to prepare the cluster for workloads. This leads to disjointed processes and difficulties in achieving a smooth end-to-end deployment experience.

The rolling upgrade process, requiring significant manual intervention, is particularly time-consuming, resulting in delays and extending the process duration to 12-15 hours. This manual intervention hinders efficiency and operational speed.

Resiliency tests have indicated that an imperative "Blackbox" approach should not handle the deployment of any add-ons. Instead, specialized software such as FluxCD/ArgoCD should be delegated to manage this task. By leveraging dedicated software, we can ensure the deployment of add-ons is separate and optimized for resilience.

To effectively manage infrastructure-related resources, Swisscom aims to delegate these responsibilities to a management cluster equipped with CAPI controllers, aligning with CAPI architecture principles. This approach involves having a management cluster that controls and reconciles workload clusters.

However, we currently observe limited capabilities to restore the state of the infrastructure after experiencing issues. The lack of robust reconciliation mechanisms often necessitates full redeployments as the only viable solution in the face of problems.
Regarding the tooling landscape, Swisscom strives to prioritize the use of upstream solutions. However, adoption of these upstream solutions is sometimes hindered by compatibility issues with Cloud Native Functions (CNFs) provided by vendors, posing challenges in achieving seamless integration.
To overcome these pain-points, Swisscom is actively working towards refining automation processes, streamlining upgrade procedures, improving resiliency strategies, and advocating for standardization and compatibility in the tooling landscape. By addressing these challenges head-on, we aim to enhance the efficiency, reliability, and overall robustness of our infrastructure deployment and management practices.

Lukas Leuthold, Swisscom

Area 3 – Observability

3 - 1 Definition

Observability is the ability to understand the state and behaviour of a system based on the outputs it produces. An observable software system provides enough information about its performance and internal state to enable users or automation to derive insights and take corrective action when needed. In the context of cloud-nativeness, observability becomes critical due to the highly distributed nature of the cloud, the complexity of modern applications, and the interactions across multiple layers of infrastructure. It is therefore essential to have an observable architecture that can pinpoint problems before they occur and support rapid remediation.
In the telecom domain, the traditional integrated node design served as the aggregated source of hardware, software, configuration, network function service delivery, and telemetry. With cloudification, this changes: the lifecycle of a network function service is decoupled from the underlying hardware and platform services. The separation of shared infrastructure, shared platform services, and dedicated application software results in telemetry being emitted independently from each disaggregated aspect of the system. There is no longer a single telecommunications node that aggregates these signals automatically, which has major consequences for 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. […]" Consequently, how observable a system is will significantly impact its operating and development costs." Translating this into the telco domain implies that telemetry data must be collected across all layers of a distributed and disaggregated application stack, processed, and made available-both to human operators and to automation systems-to fulfil multiple operational needs, such as service assurance, monitoring, closed-loop automation, and root-cause analysis. All telemetry data must also carry contextual metadata to enable meaningful interpretation.

A common cloud-native approach relies on three raw telemetry data types-often referred to as the three pillars of observability:

Metrics
Quantitative measures of system or application performance, or business-defined indicators, typically stored in time-series databases.

Logs
Structured or unstructured textual records that provide insight into runtime behaviour and context across applications and infrastructure.

Traces
Information about the path of a request across different services in a distributed architecture, enabling visibility into distributed transactions.

From these raw data types, additional insights-such as alerts and fault indications-can be derived through correlation and analysis. To implement observability at scale, it is common practice to combine appropriate tools and frameworks into an "observability stack."
A cloud-native observability approach typically includes the following concepts:

- Scraping and externalizing raw telemetry from data sources across the cloud stack.
- Providing a control plane for dynamic configuration of telemetry type and granularity.
- Storing telemetry data in a centralized, persistent data store so it can be collected once and shared across multiple consumers.
- Exposing telemetry and its schemas via interfaces that support flexible queries.
- Post-processing telemetry data (aggregation, correlation, analysis) by dedicated data consumption components to support the desired observability use cases.

3 - 2 External References

- CNCF Observability definition
- Cloud-Native Observability of Telco Apps, Ericsson
- Observability Whitepaper, v1.0 - CNCF TAG Observability, 2023
- Cloud Native Observability: Hurdles remain to understanding the health of systems

3 - 3 Pain Points (What's difficult)

The main challenge in observability is to properly define what outputs of the system or software need to be observed. Teams need to define the right metrics to monitor, generate meaningful logs and tracings. To achieve this, several teams and profiles need to collaborate such as (but not limited to): developers, DevOps engineers and product managers.
Furthermore, the significantly large and diversified landscape of observability tools result in an additional burden for organizations to handle their observability stack, with some tools lacking proper documentation or being hard to install and operate, this tendency has been reported by a micro-survey from CNCF in 2022.

Disaggregated and distributed applications
As the cloud-native telco applications are deployed in a disaggregated and distributed manner, there is no possibility to expose a network function service view from within the applications. The realizing SW components do not, and should not, have the full visibility of all layers of the cloud stack. In a similar manner, the shared cloud platforms and infrastructure are agnostic to the service functions provided by the SW components deployed on them. That leads to telemetry data streams from the independent data sources of the layers of the cloud stack which need to be aggregated and correlated to create the desired network function service view. Despite this, the requirement is still the same as pre-cloud native applications, to provide a network function service view, and the most common requirement is to do so from the application.

Ephemeral data sources
The SW components of the cloud-native telco applications are containerized and deployed on supporting container platforms. The container instances are ephemeral objects which are terminated and re-instantiated in an arbitrary way by the scheduling container control plane. The SW components which act as telemetry data sources therefore have a different lifetime than the required availability of the telemetry data itself.

Unstructured data
Many of the SW components realizing the applications and the underlying cloud stack emit telemetry data, especially logs and event information, in an unstructured format. Even though that information might be human readable, for automated machine processing of that data it is difficult to aggregate and correlate unstructured data from different sources, potentially even with different semantics behind the data.

Data volume
The sheer amount of data sources, especially when considering a complete network, leads to an immense potential volume of telemetry data which then needs to be collected and stored. The full amount of that data cannot be transported and persisted, as it will exceed the available bandwidth of the management network and the available storage capacity. Telemetry data generation and collection need to be configured and controlled to limit the volume.

Data traceability
The telemetry data scraped from the multiple data sources of the different layers of the cloud stack often lacks consistent identifiers of its sources. Even if the data objects are tagged with meta-data from their origin, the values cannot be correlated externally from their source domain. The semantics of the meta-data structure are not aligned in between the data sources.

3 - 4 Harmonized Productivity Gains Quantified

-

3 - 5 CNTF's ask to ISV or F/OSS

AREQ A3-1.001	Telemetry data should be structured: To enable consistent post-processing, all observability data should be structured according to well defined and versioned schemas.
AREQ A3-1.002	Telemetry data should be traceable to its source: Each record and metric of the generated data should be traceable in at least two dimensions: time & place. Consistent source IDs are required.
AREQ A3-1.003	Telemetry data life cycle shall be de-coupled from its source: The data needs to be available after the lifetime of its ephemeral data source.
AREQ A3-1.005	Telemetry data persistency: Observability data shall be persistently stored separately of the data source, while local caching is permitted.
AREQ A3-1.006	Telemetry data access and control: The access to the observability data to authorized users needs to be preserved. The collection and exposure of the observability data can be controlled and configured.

3 - 6 CNTF's ask to CSP readiness

AREQ A3-2.001	Ability to consume new types of telemetry data
AREQ A3-2.002	Mindset change to understand concepts and consequences of cloud-native deployments on observability and adapt or swap out southbound interfaces of today's tooling and management systems
AREQ A3-2.003	Ability to integrate new telemetry interfaces from cloud platforms and interfaces.

3 - 7 Testimonies

-

Area 4 - Scalability

4 - 1 Definition

Scalability refers to the dynamic expanding and contracting of resources as the workload fluctuates without a loss of performance or stability. It is crucial to allow the accommodation of vastly differing loads without statically overprovisioning the application. In the context of cloud nativeness this can be achieved through techniques such as auto-scaling, container orchestration, and microservices architecture. Microservice architecture breaks applications down into smaller independent units which allow different parts of the application to be scaled individually. A container orchestration platform allows the dynamic and automated management of the deployed application by performing the necessary operations, potentially across different hosts.

4 - 2 External References

-

4 - 3 Pain Points

A lack of scalability often leads to significant issues when applications face highly variable traffic. Applications may struggle to handle high traffic peaks or become severely overprovisioned during low-demand periods, resulting in performance degradation or unnecessary costs.
One major factor contributing to scalability challenges is the monolithic software architecture, which frequently causes operational issues due to its inflexibility. The slow start-up time of monolithic applications hampers the horizontal scaling process, as adding additional instances becomes time-consuming. This limitation often forces applications to resort to vertical scaling, where the computing power of a single machine is increased. Vertical scaling requires downtime, is limited by the physical properties of the server, and introduces a single point of failure.
Another issue with horizontal scaling is state management. Applications needing previous sessions or data for future transactions struggle to reroute requests to different instances of the same application. This difficulty complicates the consideration of necessary prerequisites for current requests.
The same constraints apply to downscaling; it may require downtime and is restricted by the server's physical properties. These limitations make rightsizing challenging, as the required computational capabilities must be estimated upfront and cannot be quickly adjusted, often only with downtime.
Another problem with rightsizing is that only the entire application can be upscaled. This means if only a specific part of the application is overloaded, it is still necessary to scale up even the parts that are not overloaded. These inefficiencies in rightsizing lead to cost disadvantages, as the application is likely sized larger than necessary for extended periods.

4 - 4 Harmonized Productivity Gains Quantified

-

4 - 5 CNTF's ask to ISV or F/OSS

AREQ A4-1.001	Generally, applications and their micro-services must be either stateless or externalize their state.
AREQ A4-1.002	The application must be delivered in a micro-service architecture according to CNCF and 12factor app specification
AREQ A4-1.003	The micro-services must start-up and shutdown gracefully.
AREQ A4-1.004	The application and its micro-services must support a dynamic service discovery and load balancing mechanisms
AREQ A4-1.005	The application must be deployed in Kubernetes and support its dynamic orchestration
AREQ A4-1.006	The application and its micro-services must as much as possible hardware independent.
AREQ A4-1.007	The application's micro-services must respond to autoscaling policies tailored to the specific service

4 - 6 CNTF's ask to CSP readiness

-

4 - 7 Testimonies

-

Area 5 - High Availability

5 - 1 Definition

Availability in the context of cloud nativeness typically refers to fault tolerance, resiliency and self-healing. These attributes may allow an application to stay operational even when facing errors or failures. By dynamically and swiftly addressing these issues without causing downtime, applications can maintain high availability. This is often achieved through the implementation of microservice architectures, redundancies, failover mechanisms, monitoring, and container orchestration.
A microservice architecture can facilitate resilience by allowing individual components of an application to fail without necessarily affecting others. Additionally, if the reduced start-up times of microservices would keep downtimes short, should they occur.
Redundancy and failover mechanisms are a key component to maintain availability during the restart of components. They allow to reroute traffic to a different instance of the same component while the failing one restarts. This can be done on application, microservice or infrastructure level.
Self-healing mechanisms allow a container to handle failures without manual intervention by utilizing health checks and other observability features to identify issues and trigger the according action.

5 - 2 External References

-

5 - 3 Pain Points

The current traditional approaches create some problems when aiming to achieve high availability.
Monolithic architectures for example inherently have a single point of failure. When a problem occurs in any part of the application, it will most likely cause the entire system to fail. Due to its size the restart of this application will be time consuming.
Implementing an effective failover mechanism may also be problematic. Since component isolation is most likely not supported in a general way, a failure will lead to a loss of all data related to the current transaction. This will impact the application's ability to reroute traffic to an unaffected instance of the same application.
Additionally, many applications nowadays manage state. This means that a potential failure not only risks the current transaction's data, but also the data of all previous transactions. This will impact the applications ability to return to normal operations and severely complicates the recovery mechanisms.

5 - 4 Harmonized Productivity Gains Quantified

-

5 - 5 CNTF's ask to ISV or F/OSS

AREQ A5-1.001	Generally, applications and their micro-services must be either stateless or externalize their state.
AREQ A5-1.002	If the applications are using databases, these must support backup recovery and be deployed distributed and fully redundant
AREQ A5-1.003	The application and their micro-services must be self-healing
AREQ A5-1.004	The application and its micro-services must support a dynamic service discovery and load balancing mechanisms
AREQ A5-1.005	The application must be deployed in Kubernetes and support its dynamic orchestration
AREQ A5-1.006	The application and its micro-services must as much as possible hardware independent.

5 - 6 CNTF's ask to CSP readiness

-

5 - 7 Testimonies

-