February 2025

CNTF

A Cloud Native Telco Forum Initiative

CNTF Productivity Gains

Revision 1’

Responsible Officer: Lukas Leuthold, Swisscom

CNTF

Content

0 General Information

0.1 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. While technical aspects are vital, the adoption of cloud-native principles 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.

0.2 Scope

The content applies to the CNTF initiative, started in November 2024 by Swisscom. The findings shall be actively discussed with our ISVs and IaaS Hyperscaler.

0.3 Goal

The primary goal of this document is to express the common pain-points encountered by the CNTF associates and share the expected user experience within the cloud-native context. 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 vendors should allocate resources and make improvements.

Without the insights provided in this document, there is a risk that the appeal of cloud-native design patterns might be overlooked or dismissed as a trite cliché. To counter this perception, 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.

0.4 Document History and Outlook

Rev 1'	CW11: Initial release
Rev 2	To include: 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

1 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 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, 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. Software Delivery: 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 lifecycle automation tools to streamline processes.

2. SW Lifecycle (incl. ISSU): Effective and seamless software lifecycle management is crucial for maintaining software freshness, security, and performance. Supporting in-service software and configuration upgrades (ISSU) allows for seamless updates without disrupting ongoing sessions.

3. Test Automation: 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.

4. Release Strategies: With the microservice architecture, a canary upgrade can be implemented to test new features/versions on a limited set of subscribers. This accelerates the deployment of changes into production in a more stable manner with a smaller blast radius.

5. Feedback Loop to ISV: Test and canary results will expose potential software improvements. Establishing a lean feedback channel back to the Independent Software Vendor (ISV) is crucial, avoiding unnecessary bureaucracy. Security patches or improvement releases should adhere to the Semantic Versioning paradigm.

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).

1.4 Current Situation

Swisscom	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.

1.5 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.

1.6 CNTF's High Level Architecture Requirements

AREQ A1.001	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
AREQ A1.002	Software should be delivered via OCI
AREQ A1.003	Software releases should follow Semantic Versioning
AREQ A1.004	Software should be delivered with testing tools that allows easy automation of integration testing in the Operator's environment
AREQ A1.005	Canary rollouts should be supported by the Software

1.7 Consensus of CNTF's ask

t.b.d after finding CNTF consensus

1.8 Further statements

2 Area 2 – Configuration Management

2.1 Definition

Configuration management for **cloud-native** systems across **Day 0**, **Day 1**, and **Day 2** involves processes, tools, and best practices for managing configuration, deployment and ongoing operations of applications and infrastructure 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.

Here's a breakdown of its role at each stage:

- Day 0 means IaaS readiness – that is that infrastructure facing parameters such as Pods, networks, compute resources and cluster readiness.
- Day 1 means CNF readiness – that is a network function as a distributed application is bootstrapped and commissioned to operator specific values and the configuration aspects to integrations to surrounding systems. From that moment, the CNF is observable by resource health indicators and events.
- Day 2 means service readiness – that is a collection of network functions configured to fulfill a business intent of an expected service behavior. This includes node-facing parameters that origins from BSS (such as data plans, Class of Service etc.), service-logic, regulatory and other sources. Note it does not cover user data management (HSS/UDM subscriber data provisioning).

A cloud-native approach to configuration management would be:

- Moving away from imperative read/writes based on NETCONF, CLI, SNMP to a declarative source-of-truth mastered in GitOps and synced and reconciled with K8s operators.
- Non-repetitive configuration (day 0/1 templating) should use Kubernetes ConfigMaps or Custom Resources with Schema instead of vendor/operator injecting into helm charts.
- No productized black-boxes like traditional configuration manager, design studio etc – instead the network should expose APIs and Kubernetes Operators that should be delivered as an abstraction front-end by the CNF developers instead (bottom-up) of tools that master a full set of configuration data.
- OSS systems should access the networks over the abstraction layer based on K8s CRD/Operators without the need of traditional network management systems. The logic of VNFM/VNFO/EMS and mediation is put into many operators that each cover a specific function. Closed, productized systems as being disaggregated into native K8s.
- Instead of managing a flat model of well over 1000 parameters per Node, the data schema should be split in entities that reflect repetitive use cases of the business. Repetitive configuration tasks that we miss an API-/Operator-level approach 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 Behavior 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

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) instead of declarative cloud-native approaches (configmaps) lead to fire-and-forget and higher complexity in config management. There are no APIs provided to plan and design service logics and data plans characteristics.

Even having cloud native config 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.

2.4 Current Situation

Swisscom	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).

2.5 Harmonized Productivity Gains Quantified

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

2.6 CNTF's Architecture Requirements

AREQ A2.001	APIs and/or CRD operators to configure day 0, 1, 2 parameters.

2.7 Consensus of CNTF's aks

t.b.d after finding CNTF consensus

2.8 Further statements

3 Area 3 – Cloud & Execution Environment Management

3.1 Definition

Cloud computing encompasses an end-to-end perspective that enables Infrastructure as a Service (IaaS). This holistic view includes various components such as compute management, networking, and fabrics (L2/L3), including associated gateways. It also addresses cluster lifecycle management, infrastructure programmability (Infrastructure as Code - IaaC), handling and lifecycle management of additional installed software, secrets, certificates, and license management in both the Cloud environment and the Execution Environment visible to guest applications. Furthermore, tasks such as firmware management and rollout tactics, as well as Lights-out management / KVM, are vital considerations within this framework.

To optimize tooling and reduce unnecessary complexities, it is essential for vendors to leverage upstream open-source code for components that are not the core focus of their business. Instead of maintaining separate forks for custom features, vendors should actively contribute their changes to the code base of the respective open-source projects. This approach facilitates access to publicly available documentation for integrators and operators, allowing them to leverage the broader community's expertise. Additionally, both vendors and telcos benefit from the ability to integrate Cloud Native professionals with the necessary skills and knowledge into their respective organizations.

3.2 External References

https://sylva-projects.gitlab.io/design/technical-overview/

https://gitlab.com/sylva-projects/sylva

3.3 Pain Points (What’s difficult)

Imperative Methodology: The current 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. Creating complex Low-Level Designs (LLDs) and pre-defining all the necessary information can be time-consuming and prone to errors. The current system does not facilitate seamless and efficient changes, as configurations are often spread throughout intricate LLDs, making updates burdensome and slow.

End-to-End Automation Capability: The current setup lacks comprehensive end-to-end automation capability for all stages, 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.

3.4 Current Situation

Swisscom	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.

3.5 Harmonized Productivity Gains Quantified

not yet quantified

3.6 CNTF's Architecture Requirements

AREQ A3.001	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 A3.002	Implement Kubernetes Operators to automate the management of infrastructure components, reducing the need for manual pre-definitions.
AREQ A3.003	Continuously reconcile the actual system state with the desired state defined in Git repositories
AREQ A3.004	Provide cloud native APIs to hydrate and push desired state to running state for all infrastructure components including testing
AREQ A3.005	Provide software enable CSPs to manage the lifecycle of the clusters, handle updates, and ensure that the desired state is maintained
AREQ A3.006	Usage of vanilla upstream Open-Source software instead of vendor-maintained forks

3.7 Consensus of CNTF's aks

t.b.d after finding CNTF consensus

3.8 Further statements
