



Deutsche Telekom's Cloud Automation and Orchestration based on the Kubernetes Operator Pattern

Boyan Banev, Thomas van Briel, Dunja Burek, Dr. Patrick Derckx, Christoph Hiltz, Dr. Ahmed Medhat, Kai Steuernagel, Bernard Tsai, Dr. Roland Werding

February 2026

Boyan Banev (boyan.banev@telekom.de)

Dunja Burek (dunja.burek@telekom.de)

Dr. Patrick Derckx (patrick.derckx@telekom.de)

Dr. Roland Werding (roland.werding@telekom.de)

Contents

- Contents..... 1
- Executive Summary..... 2
- Introduction 3
- Technical Overview 5
- Design Principles..... 8
- Declarative Operations Model..... 10
- Building Blocks..... 12
- Collaboration and expectations from CNF suppliers 16
- Testimonials 17
- Business Impact 18
- Bibliography 20

Executive Summary

Telecommunications operators spend millions annually on fragmented, vendor-specific automation tools that delay service launches and drive-up operational costs. The Operator-based Cloud Automation (OCA) eliminates this inefficiency by providing a unified, production-proven automation architecture that reduces deployment costs while accelerating time-to-market.

Today's telco operators juggle incompatible automation tools from multiple vendors, resulting in 6-12 month deployment cycles and duplication of integration work across projects. OCA solves this by standardizing automation on Kubernetes—the same cloud-native technology that powers Google, Amazon, and Microsoft—enabling vendors and Deutsche Telekom (DT) to collaborate through well-defined interfaces rather than proprietary ones. Unlike centralized orchestration platforms that require custom integration for each vendor's network functions, OCA uses standardized Kubernetes interfaces, reducing integration effort.

This whitepaper details OCA's architecture, design principles, and production validation across Deutsche Telekom's network. The work is part of a larger initiative within Deutsche Telekom – Horizontal Telco Cloud (HTC) [1] to harmonize the architecture on all layers of our network deployments and across our footprint.

Compared to commercial centralized orchestration platforms, the implementation of OCA reduces automation solution cost by 30% per network function, based on a preliminary 5-year total cost of ownership analysis (detailed within DT in chapter Business Impact).

OCA's declarative, API-driven architecture provides a foundation for AI-driven automation and closed-loop operations, positioning Deutsche Telekom to adopt autonomous network capabilities as they mature.

DT has already implemented OCA as a product, named OCA Solution (OCAS) as harmonized automation suite for many network functions: Telephony Application Servers (TAS), Intelligent Network (IN), SMS Gateways (SMS GW), Media Resource Function (MRF), Domain Name System (DNS), Session Border Controllers (SBC), Home Subscriber Servers (HSS), and Network Data Layer (NDL).

Several CNF suppliers have confirmed the OCA approach and are actively collaborating with DT to implement it (see chapter Testimonials for details).

Introduction

Deutsche Telekom is driving two major changes in the way we manage our telecommunications networks:

- Transition to declarative and intent-driven orchestration of telco services and applications. This is typically enabled by centralized orchestration systems. A comprehensive overview of the first large-scale implementation of this can be found in [2].
- Migration of telco workloads towards Kubernetes-based private cloud infrastructure.

Centralized orchestration platforms provide comprehensive management but require extensive custom integration, while Kubernetes offers standardization but lacks telco-specific capabilities. Deutsche Telekom implemented the Kubernetes OCA Architecture to combine the best of both approaches. The solution manages cloud-native network function (CNF) lifecycles using Kubernetes Custom Resource Definitions (CRDs) and the operator pattern (see Figure 1). This approach combines GitOps [3] declarative management with Kubernetes-native automation, enabling complex multi-vendor orchestration scenarios previously requiring proprietary platforms.

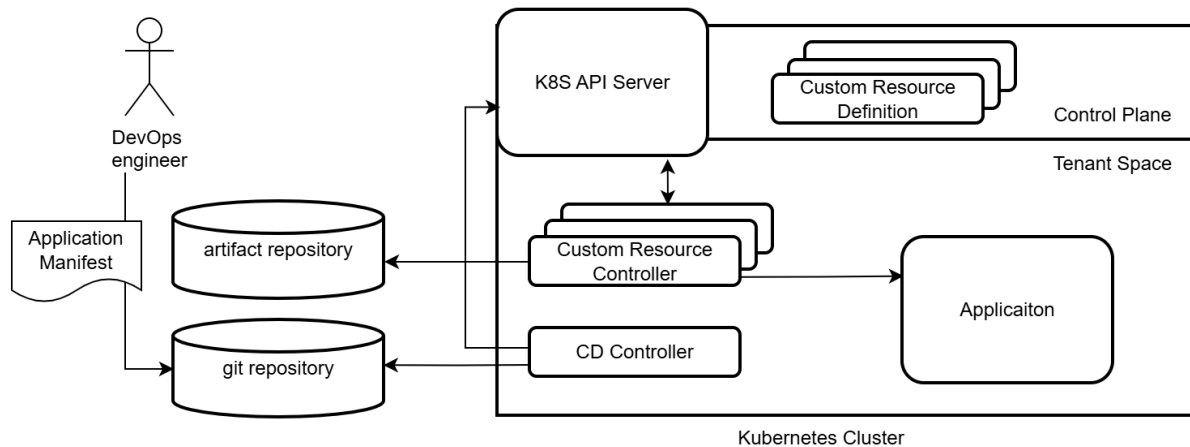


Figure 1 - GitOps and Kubernetes Operators

Since many telco applications (CNFs) are procured from suppliers, we face a fundamental automation dilemma: we want standardized, reusable automation frameworks, while network function vendors provide application-specific management tools. This creates integration conflicts and duplicated effort. OCA bridges the standardization gap by enabling vendors to provide Kubernetes-native automation while allowing DT to maintain operational control. OCA achieves this by establishing clear boundaries:

- Vendors handle CNF internal logic through their own CNF-specific Kubernetes operators.
- DT manages operational procedures and infrastructure integration through in-house-developed generic OCA operators.

The full CNF software lifecycle includes (roughly) the following steps:

1. sourcing assets from vendors
2. configuration of network functions
3. validation on premises
4. deployment/release
5. runtime operations

The OCA solutions in DT address points 2, 4, 5, and partially 3. Throughout this document the term “CNF Life Cycle Management (LCM)” is used in the context of runtime operations.

OCA addresses telco-specific challenges, which are hard to solve with standard GitOps toolset (e.g., Helm, CD tools, pipelines). These challenges come either from the CNF internal architecture and implementation or from trying to integrate cloud infrastructure with legacy OSS and BSS systems:

- statefulness of the network functions, e.g., inhibiting in-service changes
- lack of external dependency management – across clusters or towards legacy systems in the DT ecosystem.
- limited capabilities for application configuration management
- need to provision large amounts of application data
- missing service and resource inventory information
- network services with multi-vendor characteristics
- complex conditions for auto-scaling

OCA relies on GitOps and Kubernetes operator patterns to overcome these limitations. It achieves results that are on par with or better than those achieved when using commercial orchestration solutions.

Deutsche Telekom actively participates in the Linux Foundation's Nephio initiative, which aims to standardize CNF interfaces and automation. OCA complements Nephio by providing immediate automation capabilities for non-standardized vendor implementations while the industry converges on common interfaces. As vendors adopt Nephio standards, OCA operators can gradually transition to standardized interfaces, providing a pragmatic bridge to full industry standardization.

DT also actively collaborates with Swisscom within their Cloud Native Telecommunications Forum initiative.

Technical Overview

OCA leverages the Kubernetes operator pattern [4] as its core automation mechanism. Operators extend Kubernetes functionality through Custom Resource Definitions (CRDs) that define domain-specific resources, paired with controllers that implement the reconciliation logic. The CRD represents the managed object, while the controller implements the logic of how the resource is operated. In OCA, everything is modeled as a custom resource (CR).

OCA manages the following hierarchy of entities (see Figure 2):

- **CNF:** A set of containerized applications that work together to provide specific functionality or services. These containers are organized into pods, which are the smallest deployable units in Kubernetes. An application may consist of multiple pods. Each pod provides a microservice, and can use various resources like ConfigMaps, Secrets, and Persistent Volumes to manage configuration, security, and data persistence.
- **Helm Release (optional):** An instance of a Helm chart that has been deployed to a Kubernetes cluster.
- **Network Service:** A composition of multiple interconnected CNFs that collectively deliver a specific telecommunications service.

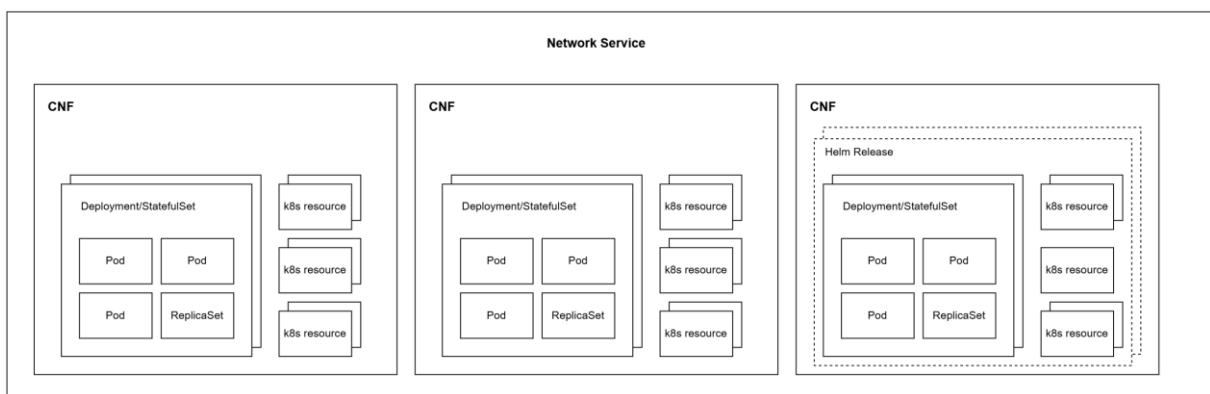


Figure 2 - CNFs and Network Services in OCA

OCA balances pragmatic implementation with rigorous architectural principles. Six core design principles guide the solution (detailed in Section "Design Principles"):

- (DP1) Declarative Lifecycle Management
- (DP2) Kubernetes-Native Automation and Operator Pattern
- (DP3) Managed Infrastructure Provisioning
- (DP4) Separation of Software, Configuration and Data
- (DP5) Modular and Hierarchical Architecture
- (DP6) Component Ownership Principle

These principles distinguish LCM operators and integration operators.

LCM operators:

- CNF LCM Operators manage the complete application lifecycle for individual Cloud-Native Network Functions, implementing vendor-specific deployment, application configuration, upgrade, scale, and termination logic and dependencies, while maintaining integration with the broader OCA framework. The CNF LCM operator manages the application configuration with Kubernetes-native resources.
- Service Operators orchestrate complex network services composed of multiple CNFs and external applications, managing inter-CNF dependencies, service-level configuration, and coordinated lifecycle operations across the entire service topology.
- Configuration Operators handle dynamic configuration management, applying runtime parameters through declarative manifests to ensure configuration consistency across CNF instances. Configuration Operators are used when Kubernetes-native resources alone cannot provide sufficient configuration management.

Integration operators provide connectivity to external telco systems including OSS/BSS platforms, inventory management systems, and workflow orchestrators, with specialized handlers for testing automation, notifications, and system integration tasks.

Figure 3 shows the different types of operators.

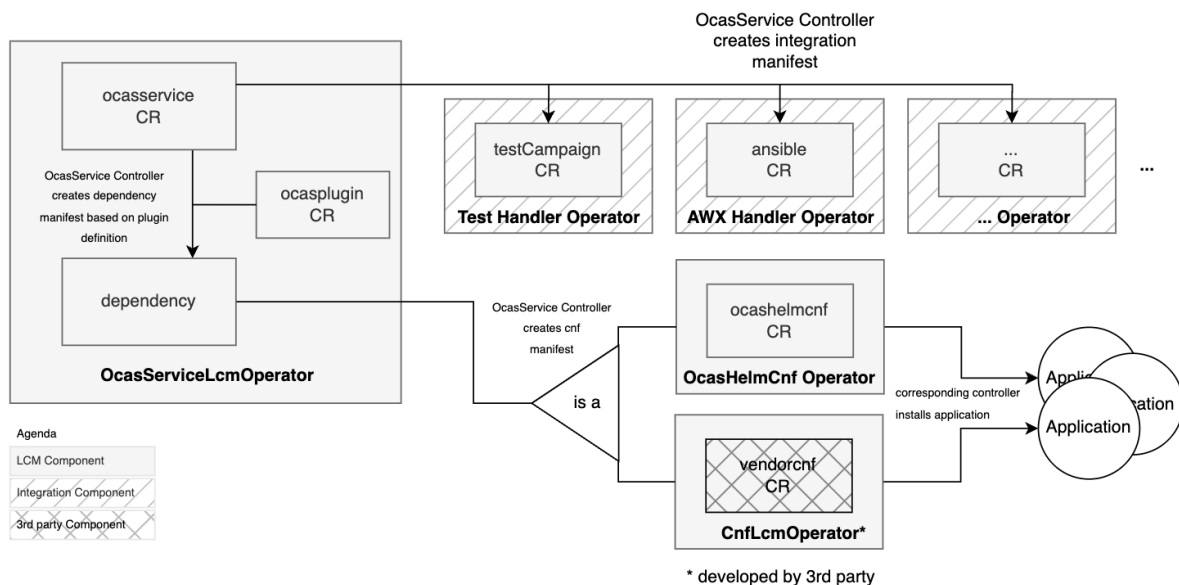


Figure 3 - Operator Decomposition

OCA supports comprehensive lifecycle management spanning from basic operations (deployment, configuration, updates, scaling, termination) through advanced scenarios including in-service upgrades and network service orchestration. Current production deployments demonstrate these capabilities. Future capabilities will include open and closed loops.

Therefore, OCA implements GitOps principles with extended scope, managing telecommunications-specific entities (CNFs and Network Services) through custom resource definitions and specialized controllers following the Kubernetes resource model.

The operation of the telco network with OCA is fully declarative and intent-driven. This means that any action that a CNF or Service operator performs is expressed as an intended state (version of resource and configuration, lifecycle state, relationships to other resources). In practice, the state is described in a text file stored in a Git repository. Any software used in the network is packaged as container images. OCA leverages proven GitOps tooling adapted for telecommunications requirements. Deutsche Telekom's implementation utilizes GitLab for repository management and Artifactory for artifact management, chosen for their enterprise scalability, security features, and integration capabilities.

OCA employs FluxCD [5] as the continuous deployment engine, selected for its GitOps-native approach, Kubernetes integration capabilities, and support for complex deployment patterns required in telecommunications environments.

Design Principles

The architecture of OCA is built around six fundamental design principles regarding the lifecycle management of CNF workloads.

DP1 Declarative Lifecycle Management: OCA uses a fully declarative approach for managing internal components and the entire application lifecycle, including deployment, configuration, updates, and termination. All operational instructions are stored version-controlled and stored in Git repositories and are expressed as desired state. This eliminates the need for imperative scripts or manual actions and establishes a foundation for GitOps workflows. Therefore, all integrations and dependencies (e.g., NS and CNF) are modeled via CRDs and configured via manifests during runtime.

DP2 Kubernetes-Native Automation and Operator Pattern: Automation in OCA is implemented directly within the Kubernetes control plane using the operator pattern and domain-specific Custom Resource Definitions (CRDs). Therefore, service orchestration and lifecycle management in OCA are managed natively, removing reliance on external orchestration software. OCA uses DT-developed operators for integration and vendor-developed operators for application automation purposes. All interactions (from lifecycle management or configuration to integrations into the DT-environment) use standard Kubernetes concepts and interfaces. This makes OCA infrastructure agnostic (as long as it is based on Kubernetes) and ensures continuous reconciliation, scalability, and observability.

DP3 Managed Infrastructure Provisioning: OCA does not handle the lifecycle of Kubernetes clusters. Nevertheless, it automates the provisioning of in-cluster infrastructure essential for CNFs and applications such as shared services, namespaces, network policies, or storage classes. These resources have lifecycles separate from applications and are established as a prerequisite for application onboarding.

DP4 Separation of Software, Configuration, and Data: OCA enforces a strict separation between software, configuration, and application data. Software components define the application's composition (e.g., container images), application data includes domain-specific resources (e.g., DNS records), and application configuration sets the application's target state (e.g., user parameters, feature flags). Therefore, in OCA each type is managed in dedicated Git repositories and applied via distinct references. This separation is essential for independent updates, clear change ownership, and minimizes error-proneness.

DP5 Modular and Hierarchical Architecture: OCA employs a modular, hierarchical design for flexibility and scalability. Components can be selected as needed based on application complexity. Simple cases use native Kubernetes resources and standard lifecycle management tools (e.g., Helm). Advanced orchestration, such as dynamic dependency management (e.g., during in-service-changes) or external integrations, is realized through layered abstractions with OCA components like service and CNF operators. This structure localizes business logic and environment-specific requirements, while higher layers coordinate cross-cutting operations and integrations.

DP6 Component Ownership Principle: Responsibility for each system component lies with the party possessing the necessary domain expertise. DT teams develop and maintain automation for platform and environment integration, while application vendors provide and support automation for their own software via CNF Operators. This allocation of responsibilities ensures that the maintenance of automation components is managed by relevant experts. This approach ensures accountability to domain competence

while separating the operating company's (DT) unique runtime environment from the vendor's application-specific installation and update logic.

Currently, component alignment (e.g., regarding requirements and capabilities) is done with documentation. Automating the exchange of component capabilities is a future development.

Declarative Operations Model

OCA leverages GitOps operational patterns (see Figure 4), specifically implementing FluxCD controllers, git-based configuration management, and declarative reconciliation loops to manage telco network functions. Therefore, OCA fully implements standard GitOps features.

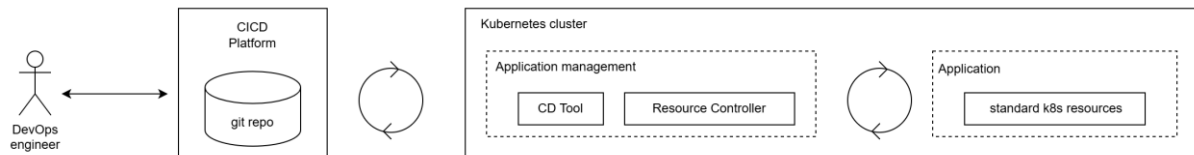


Figure 4 - GitOps Pattern with Kubernetes

OCA extends the Kubernetes resource model beyond standard workload types (pods, services, deployments) to include telco-specific resources such as CNF Custom Resources, Service Composition Resources, and Network Function State Resources. Each add-on follows standardized GitOps patterns with dedicated repository structures and automated deployment pipelines.

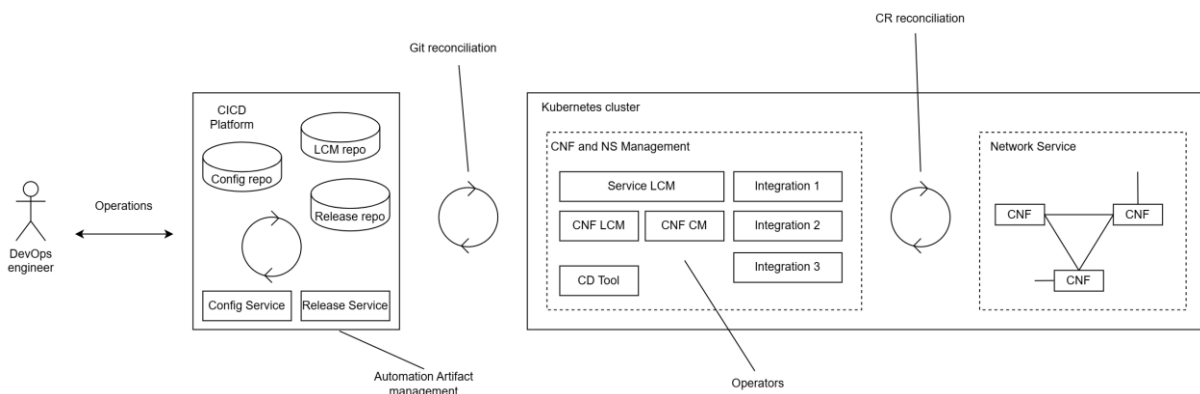


Figure 5 - Declarative Management of Network Services in OCA

OCA implements a three-tier reconciliation model: 1) Git Repository Reconciliation for version-controlled state management, 2) GitOps Controller Reconciliation where FluxCD manages deployment pipeline synchronization, and 3) Operator-Level Reconciliation where individual operators reconcile their managed Custom Resources to the desired state.

Figure 4 and Figure 5 illustrate this approach by comparing GitOps implementation with standard Kubernetes resources versus OCA extensions using Custom Resource Definitions.

Telco workload operations are initiated through Service Custom Resource (CR) manifests, which serve as the declarative interface for network service lifecycle management (see Figure 6).

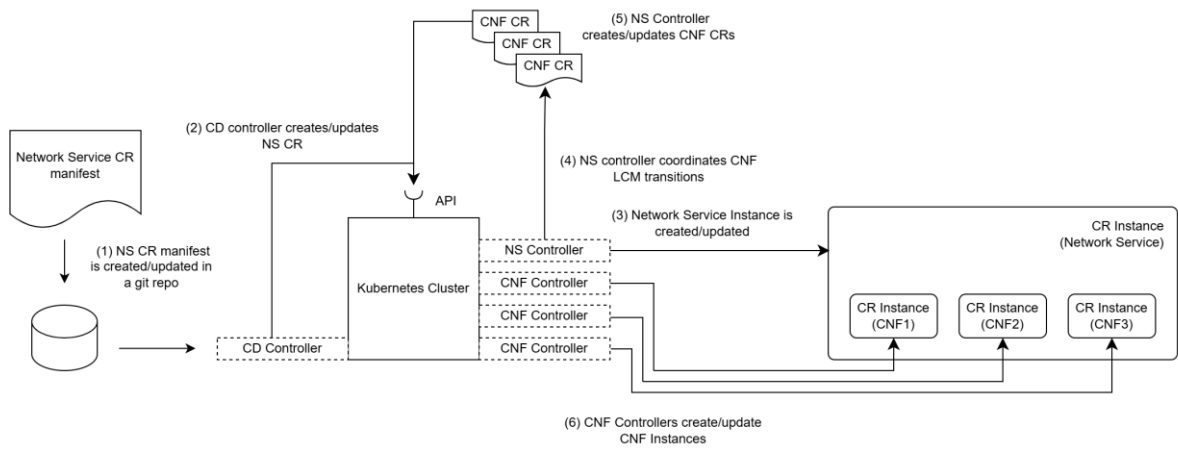


Figure 6 - NS and CNF Management in OCA

Building Blocks

OCA Lifecycle Management Operators

OCA Service Operator

The OCA Service Operator (OSO) is the central component of OCA. It deploys and orchestrates network services that can consist of multiple interconnected CNFs requiring precise sequencing and dependency management between each other or other services in the network of DT. The operator introduces the concept of a network service as a single declarative entity inside a Kubernetes cluster that encompasses multiple CNFs and their interdependencies, enabling operations teams to define complete service topologies through dependency graphs directly at the location (the cluster) where the service shall be installed.

When a Network Service definition is applied to Kubernetes, the operator analyzes the dependency structure and orchestrates component deployments in the correct sequence. All external components are modeled as CRDs as well and the operator uses the spec and status definition as the interface to the manifests.

The operator maintains comprehensive status visibility throughout the deployment lifecycle, providing real-time feedback on dependency resolution progress and enabling automated rollback capabilities when component failures are detected. Therefore, a state model has been invented that unifies the whole lifecycle of a network service (see Figure 7). The managed services move through a well-defined set of states [Created, Active, Inactive, Failed and Terminated]. Transitions between these states are triggered by events such as changes in the manifest or detected errors. The transitions are not directly (intent-driven) defined but rather chosen by the comparison of the current with the target state.

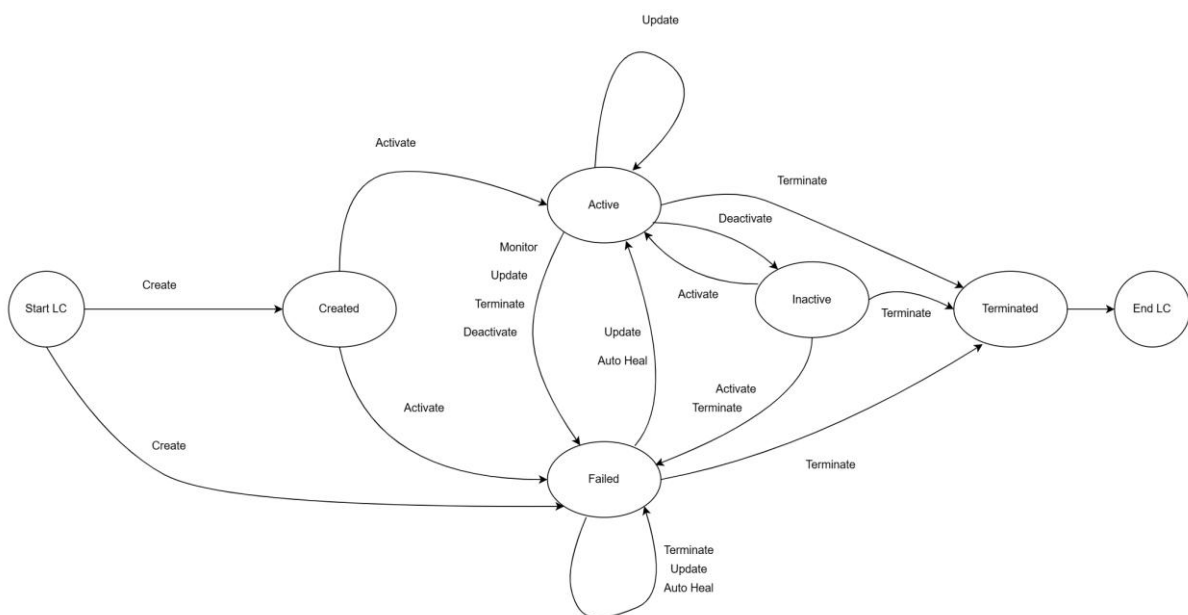


Figure 7 - OCA Service State Model

A plugin logic has been invented to apply the operator to a not-limited number of Kubernetes resources (standard and custom). In the OCA Plugin CR, the relevant state-model of the resource is modeled and

enables the OSO to control its lifecycle. Using this approach the operator follows the separation of concerns principle while rather adapting to the resource than imposing requirements for its spec or status.

CNF LCM Operators

The OCA Suite follows a separation of concerns principle where vendors are responsible for delivering automation capabilities for their own network function components. Rather than building its own pipelines or operators for all CNFs, OCA integrates vendor-delivered CNF LCM Operators that meet generic standardized requirements. This approach ensures that domain experts with deep knowledge of their network functions implement the automation logic, while OCA provides the orchestration framework and enforces consistent operational patterns across the internal telco cloud. These operators must support essential lifecycle operations (instantiate, upgrade, and teardown) while handling dependencies between internal CNF components. Critical capabilities include:

- In-Service Software Update (ISSU) that enables container image updates without service interruption, and active periodic health testing that reflects application-level service status into the CNF instance state.
- Configuration management follows cloud-native principles, with operators retrieving CNF application configurations from Git repositories or using standard Kubernetes ConfigMaps managed by FluxCD.
- Ongoing lifecycle changes are exposed through transition status fields, providing visibility into the operational state of each CNF instance.
- DT requires vendors to deliver operators that implement a simple state model for their CNFs, including Active, Created, Failed, Terminated, and Inactive states with clear transitions between them.

By establishing these requirements as integration standards, OCA ensures that vendor-delivered operators behave consistently within the overall automation framework while allowing vendors to use their expertise in implementing the lifecycle logic for their specific network functions.

For vendors that cannot deliver their own operator, OCA contains CNF Helm Operator. This operator reduces the entry requirements of using OCA to the existence of Helm Charts for each CNF. The operator orchestrates individual CNF deployments across multiple environments while maintaining strict separation between configuration, release artifacts, and infrastructure requirements. The operator implements a three-repository pattern that enables teams to manage environment-specific configurations, Helm chart releases, and infrastructure definitions independently while coordinating their integration during deployment. During deployment, the operator coordinates with Kubernetes to manage Helm chart installations, monitors resource health and readiness, and provides detailed status reporting on each deployment phase. Even if this approach is proven with several services in DT environment, the target must be to move the low-level automation logic into a vendor-delivered CNF LCM operators. The CNF Helm Operator is used as a bridge until a final solution is available.

OCA Integration Operators

The OCA Suite includes integration operators that handle aspects of DT environment automation, coordinated by the Service Operator to provide automation capabilities tailored not to a specific service but

for DT's unique environment. The following section shows the current set of integration operators used by OCA but, due to the plugin logic, it is not limited to that in the future.

The Test Automation Operator bridges the gap between deployment automation and quality assurance by integrating Deutsche Telekom's Global Test Automation Framework (GTAF) into the deployment automation. The operator automatically coordinates test campaigns that span functional validation, performance testing, security compliance, and interoperability verification as an integral part of the CNF deployment and change workflows. The tests themselves are written by domain experts and are tailored to the service itself. Quality gate functionality enables the operator to block deployment or change progression based on test results.

The Inventory Operator eliminates the operational burden of maintaining accurate asset inventories by providing automated synchronization between OCA Suite deployments and Deutsche Telekom Inventory Management Systems. The operator automatically detects deployment activities and translates them into appropriate inventory records in the form of Kubernetes manifests that are being reconciled by an inventory system. Mapping between OCA specific inventory objects modeled as CRDs (Services and CNFs) and Kubernetes basic resources are done via an annotation logic that creates hierarchies and ensures that inventory systems consistently reflect the actual state of network service deployments without manual intervention.

The Ansible Operator provides seamless integration with DT's Ansible automation platform, enabling complex workflow orchestration and operations management for services that are not only living in Kubernetes but relying e.g., on legacy integration into Virtual Machine Environments. This operator handles automated execution of operational procedures, configuration management tasks, and multi-system workflows that extend beyond the Kubernetes cluster boundary.

These integration operators work together to provide a unified automation approach, receiving coordination signals from (but not limited to) the Service Operator while maintaining their specialized functions and external system connections. Even if the Service Operator is used as the management component in a cluster to orchestrate the network service locally, all the components described before are working independently from it and can be used standalone. This modular approach enables teams to adopt integration capabilities incrementally while ensuring consistent behavior across the entire platform.

Configuration Management Tools

The Configuration Management Tools provide capabilities for generating and validating network service configurations within the OCA Suite. The Config Builder addresses the complexity of generating multi-format configurations by providing a template-based approach built on industry-standard Go-Templating engines, similar to the mechanisms used in Helm Charts. The tool extends these standard templating capabilities of Go with additional functions useful for the rendering of CNF configuration (e.g., use CSV tables as inputs, automated syntax validation for IP addresses), including enhanced processing of various file types common in network service configurations (e.g., XML for configuration and Yang schemas for validation) beyond the use of rather simple YAML files. We use the templating mechanisms to implement vendor-specific rules for a dedicated CNF configuration. The Config Validator complements this by performing schema-based validation of generated configurations, supporting both Yang schemas for network equipment configuration management and JSON schemas for cloud-native service configurations.

Sensitive data, such as certificates and passwords are handled declaratively, while ensuring that its content is available only to authorized personnel.

Collaboration and expectations from CNF suppliers

Effective collaboration between service providers and CNF suppliers determines OCA implementation success, requiring specific technical interfaces and operational agreements that differ significantly from traditional telco vendor relationships. OCA collaboration requires CNF suppliers to develop, maintain, and support dedicated Kubernetes operators that implement CNF lifecycle management while adhering to OCA architectural requirements and interface specifications. OCA adoption requires CNF suppliers to contractually commit to developing operators that meet specific functional requirements and CRD specifications (see Figure 8, Model 1), with compliance validated through defined testing and certification processes. When CNF LCM operators meet functional and CRD requirements, they provide the foundation for streamlined integration, though successful production deployment still requires comprehensive testing, operational procedure development, and ongoing collaboration between vendors and operators.

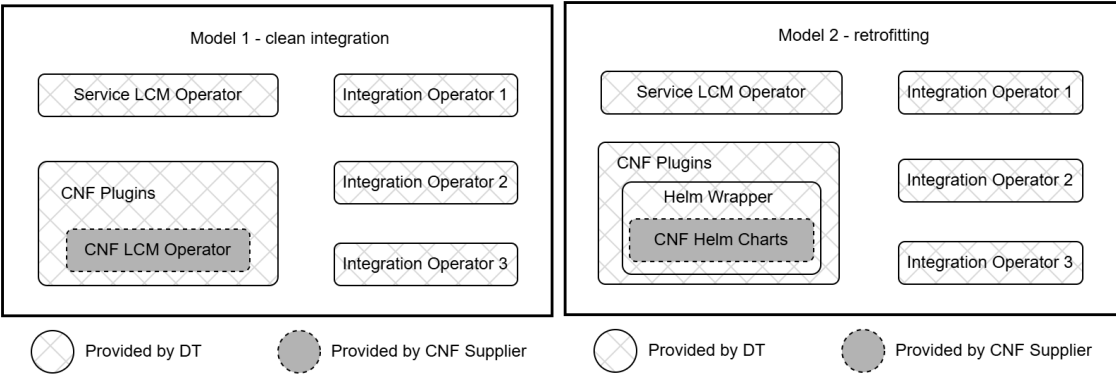


Figure 8 - DT and suppliers collaboration models

This concept is well known in the IT industry, where any application that cannot be managed by standard Kubernetes resources is complemented by a Kubernetes operator. In current projects, DT sees limited Kubernetes operator adoption among CNF vendors, with most suppliers still relying on traditional automation approaches, creating a need for transitional integration strategies. To address these limitations, OCA includes wrapper components that enable Helm chart-based CNFs to integrate with the operator-based framework, though this approach introduces additional complexity. This allows OCA to manage both the advanced as well as legacy CNFs (see Figure 8, Model 2).

In the long run DT plans to transition from Helm chart-based automation to CNF LCM operators, requiring coordinated vendor capability development, migration planning, and timeline alignment with market readiness.

Testimonials

Some of DT's key partners have confirmed the OCA approach and are actively collaborating with us to implement it.

"At Nokia, we strongly support and align with the principle that Kubernetes Operators represent the most robust and scalable approach for CNF management. This model ensures automation, resilience, and true cloud-native agility. We commend DTAG for their visionary leadership in driving OCAS - a comprehensive cloud-native automation framework built on the same principles we embrace. Together, we are shaping an ecosystem where automation and openness accelerate innovation for the entire industry."

Marcelo C. Madruga, VP Head of Portfolio Technology & Architecture, Nokia

"Titan.iium Platform is committed to the continuous deployment of cloud-native network functions, using Kubernetes operators to streamline configuration and lifecycle management. Integrating the operators with the OCAS automation tool of Deutsche Telekom delivers faster, more reliable deployments through seamless, event-driven orchestration and clear, real-time operational visibility. "

Patrik Rokyta, CTO, Titanium

"Ribbon Communications is pleased to be a key supplier and member of the ecosystem for Deutsche Telekom's operator-based OCAS solution. We're fully committed to building cloud-native products and solutions that leverage the industry's best open interfaces. It's exciting to be working with DT as they leverage GitOps methodologies and standard Kubernetes APIs to streamline lifecycle management, enabling seamless deployments, upgrades, and ongoing management across various CNF components. OCAS's innovative use of open, standardized practices is a key step toward ensuring operators can confidently manage heterogeneous CNF environments with greater agility and efficiency, benefiting our entire industry."

Steve Shusta, Sr. Director Engineering, Ribbon Communications

DT is looking forward to expanding the list of partners to collaborate with.

Business Impact

Deutsche Telekom has been implementing comprehensive automation strategies for several years. While automation is crucial for modern telco network operations, the organization maintains a cautious approach to automation development to ensure positive returns on investment. Its objective has evolved toward achieving 'Brutally Efficient Automation' - automation that maximizes operational efficiency while minimizing implementation and maintenance costs. OCA allows DT to achieve that by:

Reduced complexity: when Kubernetes is used as the automation platform, the dependency on vendor-specific software LCM automation and orchestration tools is reduced.

Straightforward technology adoption: while not yet common in the telco industry, OCA relies on technologies and principles well established in IT environments. This approach potentially broadens the pool of skilled professionals who can develop and operate network automation solutions.

Reusability: In the public space, organizations benefit significantly from sharing Kubernetes operators for popular software applications, as evidenced by the extensive operator ecosystem in the Kubernetes community [6]. DT achieves the same internally. Whenever a solution is developed for a CNF from a specific supplier, then it is immediately re-usable in other parts of the company. The larger the scale of sharing, the greater the savings.

Mutually beneficial collaboration: There have often been conflicts between Deutsche Telekom's existing automation platforms and vendor-provided automation solutions. It is not easy to draw the demarcation line between who provides which part of the automation solution. With OCA, DT shifts the disagreements towards collaboration. The CNF supplier provides the CNF LCM operator and DT provides the rest. The integration happens via a well-known API (Kubernetes) and the interfaces are based on CRDs. DT benefits by having simplified and portable automation framework, which can be used with any CNF. The CNF suppliers benefit by having a fast and easy integration in any CSP that uses GitOps and Kubernetes.

To quantify the increase in efficiency an internal analysis comparing the cost and time required to develop and maintain end-to-end CNF LCM automation solutions using centralized orchestration versus OCA approaches was conducted. The orchestrator used in the study is a commercial product, which allows DT to develop its own solutions. Preliminary results suggest cost savings with OCA compared to centralized orchestration approaches, though these findings require additional validation at a larger scale. In the comparison all activities involved in the process of delivering an automation solution were considered – technical negotiations with CNF suppliers, design and development of the LCM pipelines, testing and roll out in production. The LCM pipelines typically require maintenance for 5 years, which was considered. In real CNF projects it was found that automation solution delivery cost was 30% lower with OCA, compared to the centralized orchestration approach. The most significant saving was observed to be the time it takes to deliver the automation solution. Using OCA allowed DT to reduce this time by 73%.

Furthermore, we compared the cost of owning and managing the automation suite, which is needed to provide CNF LCM automation solutions. In the case of centralized orchestrator, this is the cost of the software licenses, professional services for integration, new features development and support. Similarly to the solutions, the cost was calculated for a period of 5 years. We calculated that owning the OCA software suite costs 67% less than an orchestrator.

Bibliography

- [1] Analysys Mason, "Deutsche Telekom's Horizontal TelCo Cloud (HTC) establishes a new industry blueprint for telco clouds," 2026.
- [2] Analysys Mason, "A move to cloud changes the game for Deutsche Telekom's next-generation IMS," 2021.
- [3] "Introduction to GitOps," [Online]. Available: <https://docs.gitops.weaveworks.org/docs/0.22.0/intro/>.
- [4] "Kubernetes Operator pattern," 16 July 2024. [Online]. Available: <https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>.
- [5] "FluxCD," Linux Foundation, 2025. [Online]. Available: <https://fluxcd.io/>.
- [6] "Operator Hub," 07 2025. [Online]. Available: <https://operatorhub.io/>.