NFVRG
Internet Draft
Intended status: Informational
Expires: April 2015

C. Meirosu
Ericsson
A. Manzalini
Telecom Italia
J. Kim
Deutsche Telekom
R. Steinert
SICS
S. Sharma
iMinds
G. Marchetto
Politecnico di Torino

October 27, 2014

# DevOps for Software-Defined Telecom Infrastructures
draft-unify-nfvrg-devops-00.txt

## Status of this Memo

## Abstract

The introduction of virtualization technologies, starting from the physical layer and going all the way up to the application plane, is transforming the telecom network infrastructure onto an agile, model-driven production environment for communication services. Carrier-grade network management was optimized for environments built with monolithic physical nodes and involves significant deployment, integration and maintenance efforts from network service providers. The DevOps movement in the data center is a source of inspiration regarding how to simplify and automate management processes for software-defined infrastructure. This first version of this draft identifies three areas that we consider key to applying DevOps principles in a telecom service provider environment, namely for monitoring, verification and troubleshooting processes. Finally, we introduce challenges associated with operationalizing DevOps principles at scale in software-defined telecom networks.

## Table of Contents

## 1. Introduction

Carrier-grade network management was developed as an incremental solution once a particular network technology matured and came to be deployed in parallel with legacy technologies. This approach requires significant integration efforts when new network services are launched. Both centralized and distributed algorithms have been developed in order to solve very specific problems related to configuration, performance or fault management. However, such algorithms consider a network that is by and large functionally static. Thus, management processes related to introducing new or maintaining functionality are complex, and costly due to significant efforts required for verification and integration.

Network virtualization, by means of Software-Defined Networking (SDN) and Network Function Virtualization (NFV), is creating an environment where network functions are no longer static and embedded into physical boxes deployed at fixed points. The virtualized network is dynamic and open to fast-paced innovation enabling efficient network management and reduction of operating cost for network operators. A significant part of network capabilities are expected to become available through interfaces that resemble the APIs widespread within datacenters instead of the traditional telecom means of management such as the Simple Network Management Protocol, Command Line Interfaces or CORBA. Such an API-based approach, combined with the programmability offered by SDN interfaces [I-D. draft-irtf-sdnrg-layer-terminology-04], open opportunities for handling infrastructure, resources, and Virtual Network Functions (VNFs) as code, employing techniques from software engineering.

The efficiency and integration of existing management techniques in virtualized and dynamic network environments are limited, however. Monitoring tools, e.g. based on simple counters, physical network taps and active probing, scale poorly and provide only a small part of the observability features required in such a dynamic environment. Huge amounts of monitoring data can be collected from the nodes, but the typical granularity is coarse-grained. Although debugging and troubleshooting techniques developed for software-defined environments are a research topic that has gathered interest in the research community in the last years, it is yet to be explored how to integrate them into an operational network management system. Moreover, tools that have been developed in academia are limited to solving very particular, well-defined problems, while they were not built for automation and integration into network operations workflows.

We acknowledge that several standardization organizations have a stake in this area. IETF working groups have activities in the area of OAM [I-D.draft-aldrin-sfc-oam-framework] and Verification [I-D.draft-lee-sfc-verification-00] for Service Function Chaining. At IRTF, the authors of [RFC7149] ask a set of relevant questions regarding operations of SDNs. The ETSI NFV ISG defines the MANO interfaces [NFVMANO], and TMForum investigates gaps between these interfaces and existing specifications in [TR228]. The need for programmatic APIs in the orchestration of compute, network and storage resources is discussed in [I-D.draft-unify-nfvrg-challenges-00].

From a research perspective, problems related to operations of software-defined networks are in part outlined in [SDNsurvey] and research referring to both cloud and software-defined networks are outlined by the EU FP7 UNIFY project in [D4.1].

The purpose of this first version of this document is to act as a discussion opener in NFVRG by describing a set of principles that are relevant for applying DevOps ideas to managing software-defined telecom network infrastructures. We identify challenges related to developing tools, interfaces and protocols that would support these principles and leverage standard APIs for simplifying management tasks.

## 2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

## 3. DevOps Principles for Software-Defined Telecom Infrastructure

In an Internet company, an agile developer is focused on releasing small iterations of their code with high velocity and high quality into a production environment. The code needs to undergo a significant amount of automated testing and verification with pre-defined templates in a realistic setting. From the point of view of infrastructure management, the verification of the network configuration as result of network policy decomposition and refinement, as well as the configuration of virtual functions, is one of the most sensitive operations. When troubleshooting the cause of unexpected behavior, high-granular visibility onto all resources supporting the virtual functions (either compute, or network-related) is paramount to facilitating fast resolution times. While compute resources are typically very well covered by debugging and profiling toolsets based on many years of advances in software engineering, programmable network resources are a still a novelty and tools exploiting their potential are scarce.

We identify two dimensions of the "developer" role in software-defined infrastructure. One dimension refers to the person that determines which high-level functions should be part of a particular service, decides what logical interconnections are needed between these blocks and defines a set of high-level constraints or goals related to parameters that define the a Service Function Chain. This person might be the product owner for a particular family of services offered by a telecom provider. They might be a key account representative that adapts an existing service template to the requirements of a particular customer by adding or removing a small number of functional entities. We refer to this person as the Service Developer and for simplicity (access control, training on technical background, etc.) we consider the role to be internal to the telecom provider. The other dimension of the "developer" role is a person that writes the software code for a new virtual network function. Depending on the actual virtual network function being developed, this person might be internal or external to the telecom provider. We refer to them as VNF Developers.

The role of an Operator in software-defined infrastructure is to ensure that the deployment processes were successful and a set of performance indicators associated to a service are met while the service is supported on virtual infrastructure within the domain of a telecom provider.

In line with the generic DevOps concept outlined in [DevOpsP], we consider that the following four principles as important for adapting DevOps ideas to software-defined infrastructure:

* Deploy with repeatable, reliable processes: Service and VNF Developers should be supported by automated build, orchestrate and deploy processes that are identical in the development, test and production environments. Such processes need to be made reliable and

trusted in the sense that they should reduce the chance of human error and provide visibility at each stage of the process, as well as have the possibility to enable manual interactions in certain key stages.

* Develop and test against production-like systems: both Service Developers and VNF Developers need to have the opportunity to verify and debug their respective code in systems that have characteristics which are very close to the production environment where the code is expected to be ultimately deployed. Customizations of Service Function Chains or VNFs could thus be released frequently to a production environment in compliance with policies set by the Operators. Adequate isolation and protection of the services active in the infrastructure from services being tested or debugged should be provided by the production environment.

* Monitor and validate operational quality: Service Developers, VNF Developers and Operators must be equipped with tools, automated as much as possible, that enable to continuously monitor the operational quality of the services deployed on software-defined infrastructure, as well as the infrastructure itself. Monitoring tools should be complemented by tools that allow verifying and validating the operational quality of the service in line with established procedures which might be standardized (for example, Y.1564 Ethernet Activation [Y1564]) or defined through best practices specific to a particular telecom operator.

* Amplify feedback loops: An integral part of the DevOps ethos is building a cross-cultural environment that bridges the cultural gap between the desire for continuous change by the Developers and the wish by the Operators for stability and reliability of the infrastructure, and feedback from customers is collected and transmitted throughout the organization. From a technical perspective, such cultural aspects could be addressed through common sets of tools and APIs that are aimed at providing a vocabulary common to Developers and Operators, as well as simplifying the reproduction of problematic situations in the development, test and operations environments.

## 4. Stability Challenges

The dimensions, dynamicity and heterogeneity of networks are growing continuously. Monitoring and managing the network behavior in order to meet technical and business objectives is becoming more and more complicated and challenging, even more when considering the need of predicting and taming potential instabilities.

In general, instability in networks may have primary effects both jeopardizing the performance and compromising an optimized use of resources, even across multiple layers: in fact, instability of end-to-end communication paths may be dependent both on the underlying transport network, as well as the higher level components specific to flow control and dynamic routing. For example, arguments for introducing advanced flow admission control are essentially derived from the observation that the network otherwise behaves in an inefficient and potentially unstable manner. Even with resources over provisioning, a network without an efficient flow admission control has instability regions that can even lead to congestion collapse in certain configurations. Another example is the instability which is characteristic of any dynamically adaptive routing system. Routing instability, which can be (informally) defined as the quick change of network reachability and topology information, has a number of possible origins, including problems with connections, router failures, high levels of congestion, software configuration errors, transient physical and data link problems, and software bugs.

As a matter of fact, the states monitored and used to implement the different control and management functions in network nodes are governed by several low-level configuration commands (today still done mostly hand-made); there are several dependencies among these states and the logic updating the states (most of which are not kept aligned automatically). Normally, high-level network goals (e.g., connectivity matrix, load-balancing, traffic engineering goals, survivability requirements, etc) are translated into low-level configuration commands (mostly hand-written) individually executed on the network elements (e.g., forwarding table, packet filters, link-scheduling weights, and queue-management parameters, as well as tunnels and NAT mappings). Network instabilities due to configuration errors can spread from node to node and propagate throughout the network.

DevOps in the data center is a source of inspiration regarding how to simplify and automate management processes for software-defined infrastructure.

As a specific example, automated configuration functions are expected to take the form of a “control loop” that monitors (i.e., measures) current states of the network, performs a computation, and then reconfigures the network. These types of functions must work correctly even in the presence of failures, variable delays in communicating with a distributed set of devices, and frequent changes in network conditions. Nevertheless cascading and nesting of automated configuration processes can lead to the emergence of non-linear network behaviors, and as such sudden instabilities (i.e. identical local dynamic can give rise to widely different global dynamics).

The CAP theorem [CAP] states that any networked shared-data system can have at most two of following three properties: 1) consistency (C) equivalent to having a single up-to-date copy of the data; 2) high availability (A) of that data (for updates); and 3) tolerance to network partitions (P). Looking at a telecom software-defined infrastructure as a distributed computational system (routing/forwarding packets can be seen as a computational problem), just two of the three CAP properties will be possible at the same time. This has profound implications technologies that need to be developed in line with the "deploy with repeatable, reliable processes" principle for configuring the states of the software-defined infrastructure. Latency or delay and partitioning properties are deeply related, and such relation becomes more important in the case of telecom service providers where Devs and Ops interact with widely distributed infrastructure. Limitations of interactions between centralized management and distributed control need to be carefully examined in such environments.

## 5. Observability Challenges

Monitoring algorithms need to operate in a scalable manner while providing the specified level of observability in the network, either for operation purposes (Ops part) or for debugging in a development phase (Dev part). We consider the following challenges:

* Scalability - relates to the granularity of network observability, computational efficiency, communication overhead, and strategic placement of monitoring functions.

* Distributed operation and information exchange between monitoring functions – monitoring functions supported by the nodes may perform specific operations (such as aggregation or filtering) locally on the collected data or within a defined data neighborhood and forward only the result to a management system. Such operation may require modifications of existing standards and development of protocols for efficient information exchange and messaging between monitoring functions. Different levels of granularity may need to be offered for the data exchanged through the interfaces, depending on the Dev or Ops role.

* Configurability and conditional observability – monitoring functions that go beyond measuring simple metrics (such as delay, or packet loss) require expressive monitoring annotation languages for describing the functionality such that it can be programmed by a controller. Monitoring algorithms implementing self-adaptive monitoring behavior relative to local network situations may employ such annotation languages to receive high-level objectives (KPIs controlling tradeoffs between accuracy and measurement frequency, for example) and conditions for varying the measurement intensity.

* Automation - includes mapping of monitoring functionality from a logical forwarding graph to virtual or physical instances executing in the infrastructure, as well as placement and re-placement of monitoring functionality for required observability coverage and configuration consistency upon updates in a dynamic network environment.

## 6. Verification Challenges

Enabling ongoing verification of code is an important goal of continuous integration as part of the data center DevOps concept. In a software-defined telecom infrastructure, service definitions, decompositions and configurations need to be expressed in machine-readable encodings. For example, configuration parameters could be expressed in terms of YANG models. It is acknowledged that the infrastructure management layers (such as Software-Defined Network Controllers and Orchestration software) might not always export such machine-readable descriptions of the runtime configuration state. In this case, the management layer itself could be expected to include a verification process that has the same challenges as the stand-alone verification processes we outline below. In that sense, verification can be considered as a set of features providing gatekeeper functions to verify both the abstract service models and the proposed resource configuration before actual instantiation on the infrastructure layer takes place.

A verification process can involve different layers of the architecture. Starting from a high-level verification of the customer input (for example, a Service Graph), the verification process could go more in depth to reflect on the service chain configuration. At the lowest layer, the verification would handle the actual set of forwarding rules and other configuration parameters associated to the service chain. This enables the verification of more quantitative properties (e.g. compliance with resource availability), as well as a more detailed and precise verification of the abovementioned topological ones. Existing verification tools for the SDN scenario could be deployed in this context, but the majority of them only operate on network configuration rules (commonly OpenFlow), and in any case all of them do not consider active network functions (i.e. VNFs or middle-boxes that dynamically change the forwarding path of a flow according to local algorithms, e.g. load balancers, packet marking modules and intrusion detection systems). Defining a set of verification tools that can account for network function virtualization is a significant challenge. In order to perform verification based on formal properties of the system, the internal states of a virtual network function would need to be represented and perhaps summarized in a way that allows for the verification process to finish within a reasonable time interval.

## 7. Troubleshooting Challenges

One of the problems brought up by the complexity introduced by NFV and SDN is pinpointing the cause of a failure in an infrastructure that is under continuous change. Developing an agile and low-maintenance debugging mechanism for an architecture that is comprised of multiple layers and discrete components is a particularly challenging task to carry out. Verification, observability, and probe-based tools are key to troubleshooting processes, regardless whether they are followed by Dev or Ops personnel.

* Automated troubleshooting workflows

Failure is a frequently occurring event in network operation. Therefore, it is crucial to monitor components of the system periodically. Moreover, the troubleshooting system should search for the cause automatically in the case of failure. If the system follows a multi-layered architecture, monitoring and debugging actions should be performed on components from the topmost layer to the bottom layer in a chain. Likewise, the result of operations should be notified in reverse order. In this regard, one should be able to define monitoring and debugging actions through a common interface that employs layer hopping logic. Besides, this interface should allow fine-grained and automatic on-demand control for the integration of other monitoring and verification mechanisms and tools.

* Troubleshooting with active measurement methods

Besides detecting network changes based on passively collected information, active probes into delay, network utilization, loss rate are important to debug errors and to evaluate the performance of network elements. While tools that are effective in determining such conditions for particular technologies were defined by IETF and other standardization organization, their use requires a significant amount of manual labor in terms of both configuration and interpretation of the results. In contrasts, methods that test and debug networks systematically based on models generated from the router configuration, router interface tables or forwarding tables, would significantly simplify management. They could be made usable by Dev personnel that have little expertise on diagnosing network defects. Such tools naturally lend themselves to integration into complex troubleshooting workflows that could be generated automatically based on the description of a particular service chain. However, there are scalability challenges associated with deploying such tools in a network. Some tools may poll each networking device for the forwarding table information to calculate the minimum number of test packets to be transmitted in the network. Therefore, as the network size and the forwarding table size increases, forwarding table updates for the tools may put a non-negligible load in the network.

## 8. Security Considerations

TBD

## 9. IANA Considerations

This memo includes no request to IANA.

## 10. References

### 10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 10.2. Informative References

[NFVMANO] ETSI, "Network Function Virtualization (NFV) Management and Orchestration V0.6.1 (draft)", Jul. 2014

[I-D.draft-aldrin-sfc-oam-framework] S. Aldrin, R. Pignataro, N. Akiya. “Service Function Chaining Operations, Administration and Maintenance Framework”, draft-aldrin-sfc-oam-framework-00, (work in progress), July 2014.

[I-D.draft-lee-sfc-verification-00] S. Lee and M. Shin. “Service Function Chaining Verification”, draft-lee-sfc-verification-00, (work in progress), February 2014.

[I-D. draft-irtf-sdnrg-layer-terminology-04] E. Haleplidis (Ed.), K. Pentikousis (Ed.), S. Denazis, J. Hadi Salim, D. Meyer, and O. Koufopavlou, “SDN Layers and Architecture Terminology”, Internet Draft, draft-haleplidis-sdnrg-layer-terminology-04 (work in progress), October 2014

[RFC7149] M. Boucadair, C Jaquenet. "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, March 2014.

[TR228] TMForum Gap Analysis Related to MANO Work. TR228, May 2014

[I-D.draft-unify-nfvrg-challenges-00] R. Szabo et al. “Unifying Carrier and Cloud Networks: Problem Statement and Challenges”, draft-unify-nfvrg-challenges-00 (work in progress), October 2014

[D4.1] W. John et al. D4.1 Initial requirements for the SP-DevOps concept, universal node capabilities and proposed tools, August 2014.

[SDNsurvey] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig. "Software-Defined Networking: A Comprehensive Survey." To appear in proceedings of the IEEE, 2015.

[DevOpsP] “DevOps, the IBM Approach” 2013. [Online].

[Y1564] ITU-R Recommendation Y.1564: Ethernet service activation test methodology, March 2011

[CAP] E. Brewer, “CAP twelve years later: How the "rules" have changed”, IEEE Computer, vol.45, no.2, pp.23,29, Feb. 2012.

## 11. Acknowledgments

This work is supported by FP7 UNIFY, a research project partially funded by the European Community under the Seventh Framework Program (grant agreement no. 619609). The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

We would like to thank in particular the UNIFY WP4 contributors, the internal reviewers of the UNIFY WP4 deliverables and Konstantinos Pentikousis from EICT, for the useful discussions and insightful comments.

This document was prepared using 2-Word-v2.0.template.dot.

## Authors' Addresses

Catalin Meirosu
Ericsson Research
S-16480 Stockholm, Sweden
Email: catalin.meirosu@ericsson.com

Antonio Manzalini
Telecom Italia
Via Reiss Romoli, 274
10148 - Torino, Italy
Email: antonio.manzalini@telecomitalia.it

Juhoon Kim
Deutsche Telekom AG
Winterfeldtstr. 21
10781 Berlin, Germany
Email: J.Kim@telekom.de

Rebecca Steinert
SICS Swedish ICT AB
Box 1263, SE-16429 Kista, Sweden
Email: rebste@sics.se

Sachin Sharma
Ghent University-iMinds
Research group IBCN - Department of Information Technology
Zuiderpoort Office Park, Blok C0
Gaston Crommenlaan 8 bus 201
B-9050 Gent, Belgium
Email: sachin.sharma@intec.ugent.be

Guido Marchetto
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 - Torino, Italy
Email: guido.marchetto@polito.it