# Roles of DevOps tools in an automated, dynamic service creation architecture

Felicián Németh[1], Rebecca Steinert[2], Per Kreuger[2], Pontus Sköldström[3],

[1]MTA-BME Information Systems Research Group, *Hungary*; [2]SICS Swedish ICT, *Sweden*; [3]Acreo Swedish ICT AB, *Sweden*

***Abstract*—Software Defined Networking (SDN) and Network Functions Virtualization facilitate, with their advanced programmability features, the design of automated dynamic service creation platforms. Applying DevOps principles to service design can further reduce service creation times and support continuous operation. Monitoring, troubleshooting, and other DevOps tools can have different roles within virtualised networks, depending on virtualization level, type of instantiation, and user intent. We have implemented and integrated four key DevOps tools that are useful in their own right, but showcase also an integrated scenario, where they form the basis for a more complete and realistic DevOps toolkit. The current set of tools include a message bus, a rudimentary configuration tool, a probabilistic congestion detector, and a watchpoint mechanism. The demo also presents potential roles and use-cases for the tools.**

## I. Introduction

The UNIFY project (www.fp7-unify.eu) aims at designing a programmability framework for flexible and dynamic operations of Virtual Network Functions (VNFs) on software-defined infrastructure [1]. It explores the building blocks necessary to enable dynamic and flexible service-chaining, where the VNFs are implemented and operated following development and operation cycles similar to those in existing production environments. Part of the work in UNIFY therefore include development of network management tools supporting *Service Provider DevOps processes* [2], such as VNF deployment, verification, troubleshooting, and monitoring processes.

Dynamic and flexible service-chaining can only be realised through a high degree of automation which need to be inherently embedded into existing and future SDN management frameworks. An example of an automated VNF deployment process include steps such as verification and testing to ensure resource availability and configuration, as well as troubleshooting when necessary. For operation, examples include automatic mitigation of performance degradations for the purpose of maintaining QoS, or re-scaling of virtual resources under varying network conditions and service usage. Supporting such management actions require methods that can provide high network observability at varying levels of detail (depending on use case) while operating in a scalable manner, and that can be activated and deployed from anywhere in the network.

The demo presented here exemplifies two perspectives of network observability and the interaction between selected monitoring, debugging, and control messaging components of the UNIFY architecture. It shows how detection of link congestion triggers automated load balancing, and how the information about the observed state and associated actions are presented to the VNF developer and operator. The information forwarded can be used both for supporting continuous service development or for supporting operator management actions.

## II. Architectural overview

The architecture proposed by UNIFY consists of three layers, enabling programmability at different abstraction levels: The *Service layer* maps abstract service requests described as a graphs of smaller service components into to graphs of deployable VNFs. Service management, e.g., SLA verification belongs to this layer as well. Based on a global network and resource view, the *Orchestrator layer* is responsible for mapping service requests to available resources and performs real-time optimisation. The *Infrastructure layer* instantiates VNFs to its physical and virtual resources including compute, storage and networking resources, and to components required for local resource management. Monitoring information are necessary at every level in this architecture, and our prototype implements key elements required to provide each layer with a suitable view of the state of the network.

Even in a virtualised networking setting, many monitoring functions need to be deployed on the forwarding nodes, and we argue that the data produced by measurements also needs to be at least partially processed and condensed locally to achieve scalability. Other observability data needs to be propagated to and combined in control plane components in more central locations before decisions or action can be taken. We thus need to supply developers and operators with the means to *specify* and *deploy* monitoring functions, and to *communicate*, *analyse* and *aggregate* the resulting information within several parts of the network. We have prototyped the following four components as examples on how these needs could be covered in the UNIFY architecture.

*A Message bus*: We've extended the ØMQ library to a lightweight, programmable, hierarchical *messaging bus* designed for high throughput and low latency scenarios, and which supports both centralized broker architectures as well as broker-less ones, and here used for communicating configuration, signalling, and measurement results between the monitoring and troubleshooting components within and between layers in the UNIFY architecture.

*A configuration tool*: Configuration of monitoring functions is an integral part of the programmability framework, and to support this aspect of the framework, we've implemented a rudimentary interpreter for a domain specific language in which we specify which measurement functions should be activated and when, what and where they should measure, how to further process the produced data, and what actions to take based on the results.

*A congestion detector:* As an example of a specific monitoring function, we demonstrated a scalable probabilistic congestion detector [3] based on the analysis of the rate distribution on individual links at different time scales. Querying counter statistics locally at high rates allows for accurately capturing

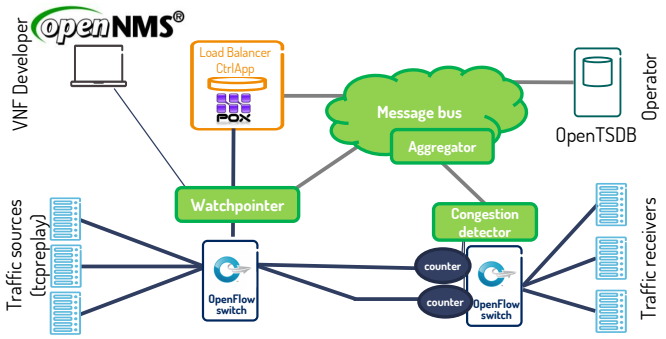


Fig. 1. The demo setup containing the four DevOps tools. (The configuration tool is responsible for initialising the Aggregator component.)

important aspects the traffic behaviour with significantly lower overhead than in a centralized setting, and by varying the querying rate we can achieve flexible high quality monitoring without the cost of constant high rate sampling of the counters. The counter values are used for estimating the parameters of the rate distribution which, is in this instance, is used for detecting increased risk of link overload by inspecting rate distribution percentiles.

*A watchpoint mechanism* that intercepts the OpenFlow control channel between the controller and switches looking for packet_in messages. If a packet matches a user-defined criteria, then different actions can be triggered, e.g., dropping the message, sending alarms, or creating switch-state snapshots.

## III. Observability Aspects

In a service chaining architecture like UNIFY, monitoring, troubleshooting, and messaging components can serve several distinct purposes. They can be part of the infrastructure layer of the programmability framework and observe physical resources, e.g., link or node status; or they can be part of a service chain and instantiated as VNFs for observing and controlling service-level properties, e.g., SLA and resource usage monitoring, or as input to autonomous reconfiguration.

Moreover, the very same component can be placed in the infrastructure layer or the service chain based on users' intent. For instance, the scalable monitoring component should be part of the infrastructure if it used for link congestion detection, but it should be instantiated as a VNF if it measures the traffic intensity for one particular service chain.

Finally, the users of the monitoring information can be VNF developers or service operators. For example, a VNF developer can use the watchpoint to detect bugs in her code while testing a new VNF implementation, but later when the VNF is deployed in production the watchpoint can be used to detect configuration errors and notify an operator or an Operations Support System. Our prototype implementation is flexible enough to support all these aspects.

## IV. Demo

In the demo scenario as shown in Fig. 1, a Tier 1 ISP is providing network transit for three smaller ISPs, over its backbone SDN network. To forward the transit traffic the Tier 1 ISP has two links available, one low-latency, reliable link and a secondary link with higher latency and low reliability. Due to its SLA the third of the smaller ISP should never be routed on the secondary link whereas the other two ISPs should use the first link unless there is a significant risk of congestion, in which case the secondary link is preferred. The Tier 1 ISP additionally wants to keep its monitoring and messaging overhead low and avoid transferring data to central points in its network unless necessary.

To take link congestion risk into account, the Tier 1 ISP deploys rate monitors on the links, which calculate an estimate every second and send it to their local aggregation point. The local aggregation point has been dynamically programmed in the setup phase to apply hysteresis to the overload risk estimates and compare the results to a threshold, which if breached will cause a signal to be sent to the load balancing application on the SDN network controller. Additionally, the estimates are also sent to an OpenTSDB instance. To ensure that policies are not violated the ISP has deployed a watchpoint beneath the SDN controller, which monitors all OpenFlow control traffic and stops any control message which would cause a violation of the policies. To keep network overhead low; monitoring results, triggers, and configuration data are sent over a hierarchical distributed messaging system which forwards messages locally if possible.

The demo shows three data flows arriving in sequence to the Tier 1 network, from the different ISP, which are allocated to the primary or secondary links depending on the current overload risk. The first flow, arriving to an empty network, is placed on the primary link. The second flow, is placed on the secondary link if the primary link is above a 1% risk of overload. The third flow should always be placed on the first link (due to the SLA), but due to a bug in the load balancing software causing it to ignore this policy it will be assigned to the secondary link if the overload risk on the primary exceeds 1%. This bug however will be caught by the watchpoint which will block the traffic assignment and raise an alarm in the NMS.

The demo is running in a virtualised environment based on Mininet in combination with POX which provides the SDN network and load balancing functionality. Other open source projects used are OpenTSDB, OpenNMS, tcpreplay, and ØMQ for the messaging system. The monitoring functions, messaging bus, and aggregation are written in Python.[1]

## V. Concluding Remarks

We demonstrate an integrated prototype of several key elements in a monitoring framework for virtualised networks. This illustrates how relations and communication between components and levels of an overall architecture could be supported. The components will continue to be developed within the UNIFY project (e.g., with advanced conflict resolution in case of policy violation), and should also be of more general interest (e.g., we plan to release the extensions of the ØMQ library as open source). Finally, the congestion detector is an example of a very general type of mechanism.

## References

[1] A. Császár, W. John *et al.*, "Unifying cloud and carrier network," in *Proceedings of the Workshop on Distributed Cloud Computing (DCC 2013)*, December 2013, invited Paper, Dresden, Germany.

[2] W. John, K. Pentikousis *et al.*, "Research directions in network service chaining," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. IEEE, 2013, pp. 1–7.

[3] P. Kreuger and R. Steinert, "Scalable in-network rate monitoring," in *Proc. of IM 2015*, Ottawa, Canada, May 2015.

[1] mininet.org, www.noxrepo.org/pox, opentsdb.net, www.opennms.org, tcpreplay.synfin.net, zeromq.org