NETWORK PROGRAMMABILITY IN CLOUD-NATIVE NFV

tail-f a Cisco company

Table of Contents

1. Introduction

In the telecommunications industry, major technology evolutions have become a way of life. Just a few years ago, virtualization was the name of the game. In an effort to increase flexibility, lower costs, and accelerate time to market, operators began moving away from network devices tied to dedicated physical appliances in favor of network functions virtualization (NFV). Taking their cues from operators, network equipment providers (NEPs) created a new generation of virtualized network functions (VNFs) implemented in software, which can run in virtual machines (VMs) on commercial off-the-shelf (COTS) servers. Today, another major evolution is under way: the shift to cloud-native applications.

In a cloud-native world, applications are decomposed into "microservices" running in containers rather than dedicated VMs, so they can more easily take advantage of the shared resources, speed, and agility of cloud environments. Since VNFs are, at their core, software applications themselves, they too are now being decomposed into their constituent microservices to become "cloud-native." However, deploying and managing VNFs, especially in complex multivendor operator environments, is different than running other types of applications in the cloud.

Due to the unique requirements of the software-defined networking (SDN) automation running in service provider networks, operators (and their service orchestration systems) still require the ability to centrally manage device configurations at runtime. Therefore, VNFs will still need to be "programmable" via SDN controllers and conventional service orchestrators. For these reasons, it is just as important as ever for NEPs to ensure that the VNFs they create are programmable, even as they embrace cloud-native development methodologies.

This paper discusses some of the requirements for developing cloud-native VNFs. It details the impact of cloud-native approaches on today's programmable networks, and it describes why NEPs should continue to prioritize programmability in their VNFs for the foreseeable future.

2. Background

To understand the role of network programmability in cloud-native environments, it's important to understand how we got here. A decade ago, as communications service providers struggled to keep pace with their customers' insatiable demands for network services, their network infrastructures grew to contain a vast, constantly growing variety of proprietary hardware. This complexity had a significant impact on operator costs, efficiencies, and time to market. Launching any new services, for example, typically demanded a major network reconfiguration effort, as well as onsite installation of new equipment—which, in turn, required additional floor space, power, and trained maintenance staff. Enter NFV.

The network architecture concept behind network functions virtualization uses the technologies of IT virtualization to virtualize entire classes of network node functions. Once virtualized, these network functions can act as flexible building blocks that operators can connect, or chain together, to create communications services. By shifting away from custom hardware appliances, and using VNF software running on standard COTS servers, operators can lower costs, streamline operations, and bring up new services much more quickly.

Unfolding at the same time as NFV, and closely related to it, software-defined networking arose to support many of the same goals. By separating the control plane and data plane of their infrastructures, operators gain the ability to automate networkwide device configurations, along with the flexibility to distribute and scale data-plane capacity as business requirements dictate. SDN and NFV are complementary concepts, but increasingly codependent for network operators looking to fully realize the benefits of automation, reduced complexity, and speed.

According to Cisco's latest Global Cloud Index Report, published in February 2018, more than two-thirds of all data centers will fully or partially adopt SDN by 2021, compared to just 16 percent in 2016. But the report reveals an even bigger shift happening in the world's IT environments: the move to cloud.

The findings show that by 2021, 94 percent of all workloads will be cloud-based, and global cloud traffic will represent 95 percent of total data center traffic. The report also forecasts that SDN and NFV will carry over half of "within data center traffic"(that is, traffic that remains within the facility) over the same period, compared to 28 percent in 2016. As these projections show, while businesses in practically every industry embrace cloud, SDN and NFV will continue to play important roles in automating large data centers.

How Cloud Affects SDN and NFV

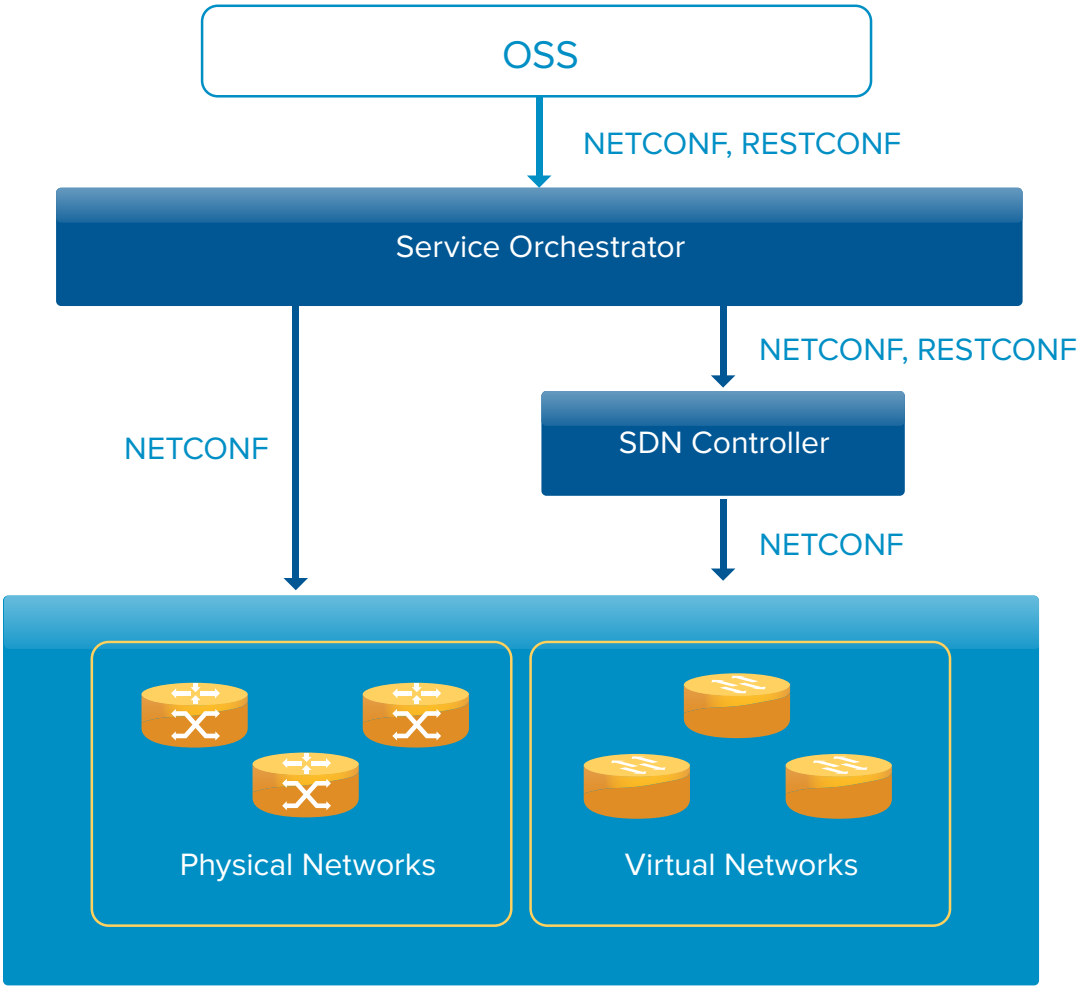
To date, the majority of NFV development efforts have focused on porting the monolithic software applications that used to live in custom hardware appliances to virtual machines (VM). However, a trend has been emerging in the world of SDN towards running NFVs in cloud-native networks.

In theory, virtualized network device software should be able to take advantage of the same benefits as any other application in the cloud: ready access to existing cloud infrastructure and resources, and much quicker time to market. In practice, however, monolithic custom hardware-based network function software applications don't scale well in distributed cloud environments. As a result, many NEPs are now reevaluating their VNF applications, looking to redesign and rewrite them as a set of distributed microservices hosted inside of containers, which can scale horizontally and run in the cloud.

Crucially, however, NEPs need to recognize that virtualized networking applications—and the SDN architectures that automate their programming—can have unique requirements that separate them from other types of applications that are being redesigned to be cloud-native. Additionally, while the industry works to migrate NFV towards cloud-native SDN environments in the data center, this process will not happen all at once. Operator networks in particular are likely to consist of "hybrid" environments, containing both cloud-native and traditional VNFs, for the foreseeable future.

What does all this mean then with regard to network programmability? Operators will continue to use traditional SDN controllers and service orchestrators to configure (or "program") downstream VNFs. Today, northbound SDN controller interfaces typically use either the NETCONF or RESTCONF protocol to communicate with service orchestrators, while service orchestrators themselves (such as Cisco® Network Services Orchestrator, or NSO), commonly use RESTCONF to communicate with an OSS. For southbound configuration of virtualized network devices, NETCONF has been widely adopted as the protocol of choice. For NEPs then, building programmable VNFs—that is, building support for NETCONF into their virtualized device software—will still be essential for those VNFs to function in real-world service provider environments.

Programmable Interfaces in a SDN Environment



3. Cloud-Native Application Development: Microservices and Containers

We will discuss the role of network programmability and centralized configuration management in cloud environments in more detail later in this paper. First though, let's review the ways that cloud-native software applications differ from traditional ones. Specifically, that they are decomposed into microservices and deployed using containers rather than dedicated VMs.

The Rise of Microservices

"Microservices" is a new architectural approach to developing an application as a loose collection of fine-grained services, rather than as a single, monolithic piece of code. In this approach, each distinct "service" implements business capabilities, runs in its own process, and communicates with other services and the cloud via HTTP APIs or messaging.

It is important to note that the reason why applications built on microservices are API-driven is to allow them to be programmable. Each microservice can be deployed, upgraded, scaled, and restarted independent of other services in the application, typically as part of an automated system. This model allows applications to be updated more frequently and easily, without impacting end customers.

Containers and Their Benefits

A "container" is a packaging format for a unit of software that ships together. Containers encapsulate a set of software and its dependencies—that is, the minimal set of runtime resources the software needs to perform its function. Like VMs, containers are, at the end of the day, ultimately just a virtualized deployment option. Containers differ from VMs, however, in some important ways.

VMs encapsulate functionality in the form of the full application platform and its dependencies, whereas containers are built for lightweight microservices. Where each VM in an environment has its own full-sized OS, containers typically have a more minimal OS. As a result, containers are typically an order of magnitude or two smaller than VMs.

Because they are lightweight and have a minimal OS component, containers have some important advantages over traditional VMs. First, they start up very quickly. Containers are more like processes, whereas VMs are more like physical machines. Containers can also move easily from one platform to another compatible one, and a number of containers can fit into the disk footprint of a single VM. As far as mobility is concerned, and the duration of maintenance windows, containers can be spun up much more quickly than VMs. Containers therefore shrink the required maintenance window significantly when compared to VMs.

In addition, immutable container images can be created at build/release time rather than at deployment time. This is because, in cloud-native environments based on microservices, each application doesn't need to be composed with the rest of the application stack, nor married to the production infrastructure environment. This capability is central to the broader appeal of cloud-native applications in general. With the ability to generate container images at build/release time, operators can maintain a single, consistent application environment from development through production. This dramatically accelerates the release of new applications, features, and updates, and enables the "continuous integration/continuous delivery" (CI/CD) cycle that is at the core of modern DevOps approaches.

It should be noted that, while containerized applications do require orchestration, this is not the same as traditional "service orchestrators" in operator networks. Rather, cloud-native application environments use orchestration software such as Kubernetes to automate the deployment, scaling, and management of containers and microservices across clusters of hosts.

4. Understanding Cloud-Native Application Development

We've been discussing the concept of "cloud-native" applications extensively, but what does the term actually mean? The cloud-native microservices container architecture originated with the web-scale providers such as Amazon, Google, and Netflix. As these massive businesses live and breathe cloud, they sought an approach to building and running enterprise applications that would fully exploit one of the central advantages of the cloud computing delivery model: on-demand computing power.

The web-scale providers spearheaded the emergence of cloud-native applications that are developed specifically for cloud platforms, built to natively utilize the infrastructure services provided by cloud computing providers. The benefits they realized were significant. Chief among them: continuous delivery. By developing applications as lightweight services tuned specifically for on-demand cloud environments, these companies were able to ship software much more quickly, and radically reduce the number of steps in the traditional build/test/release/deploy software lifecycle.

The industry at large has paid close attention, and today, organizations around the world are adopting the same CI/CD model. By developing applications as cloud-native and shortcutting the traditional software lifecycle, organizations can more effectively operate and scale applications. Most important, they gain agility: the ability to quickly add new functionality to software, even as it remains stable and secure in production.

Inside Cloud-Native Applications

For the most part, a cloud-native approach implies building applications that are assembled as a set of microservices that run inside of containers hosted in a Linux environment. This is a very different architecture than that used in traditional enterprise application design, as getting software to work in the cloud requires a broad set of components that work together.

As defined by the Cloud Native Computing Foundation (https://www.cncf.io), cloud-native architected systems are:

- **Containerized:** Each part of the system (applications, processes, and other components) is packaged in its own container. This facilitates reproducibility, transparency, and resource isolation.
- **Dynamically orchestrated:** Containers are actively scheduled and managed to optimize resource utilization.
- **Microservices-oriented:** Applications are segmented into microservices. This significantly increases the overall agility and maintainability of applications.

5. Benefits of Being Cloud-Native

There are many benefits to running cloud-native applications in a cloud computing environment. The most significant is that common infrastructure services such as high availability, scalability, and upgradability are all already provided by the cloud; it is not necessary for application developers to recreate this logic for each application. Instead, developers can focus on developing the unique business logic of the application and implementing it as a set of distributed and reusable components via microservices.

A cloud-native approach to the software delivery lifecycle fully automates the infrastructure, developer middleware, and backing services used to run applications. By employing a cloud-native approach, developers can more effectively operate, scale, and update applications, even in production. The result is applications that are continually improved by adding and refining functionality, and the ability to bring new software and capabilities to market much more quickly.

It is also worth noting that, in addition to facilitating faster application design and deployment, cloud-native development allows organizations to do more with fewer resources. Today, modestly funded startups with a few tens of engineers can build software systems in the cloud that deliver rapidly-evolving, fast-scaling services that attract tens and even hundreds of millions of users.

6. Cloud-Native NFV

Now that we've discussed cloud-native applications, we can pose the key question: how do cloud-native approaches impact NFV? In theory, NEPs ought to be able to create cloud-native VNFs the same way that organizations create any other type of cloud-native software.

At its core, cloud-native NFV implies virtual network functions that are decomposed into microservices, which are scheduled to run in containers using available compute and storage resources in the cloud computing environment, based on system demand.

Rather than capturing the functionality of a network node within a complex, monolithic piece of software, VNFs are decomposed into simple, loosely-coupled components or microservices that communicate with each other via well-defined APIs. This allows the VNFs to easily scale out horizontally and provide full redundancy.

As we will see, however, this cloud-native approach gets complicated in operator networks, where service orchestrators and SDN controllers rely on the ability to centrally push out configurations to downstream devices at runtime.

7. Cloud-Native VNF Design principles

When building cloud-native VNFs, what are the specific design considerations that NEPs need to consider? The Twelve-Factor App (https://12factor.net) has been widely used as the methodology for building cloud-native software applications that run as a service. Although all twelve factors described in the methodology are important, three of them are particularly relevant to cloud-native VNFs. They are:

- Processes: Execute the application as one or more stateless processes.
- Concurrency: Scale out via the process model.
- Config: Store configurations in the environment.

Let's explore each of these factors in more detail.

Processes

To meet the Processes requirement, the entire application should be executed as one or more stateless processes. In the case of a cloud-native VNF application, those stateless processes will be run as microservices, and any data that needs to persist must be stored in a stateful backing service, typically a database.

This represents a departure from traditional VNF development, where a single transaction could span multiple sessions, and all information was cached. That approach becomes problematic in a cloud-native world, where transactions that span multiple sessions make it difficult to scale up processing power for a particular process. Developers can still use single-transaction caching in cloud-native VNFs, but they can no longer assume that the cached information will be available for a future request or job.

Concurrency

In terms of concurrency, the processes (or microservices) should be architected with different process types to handle different workloads. For example, HTTP requests may be handled by a web process, while long-running background tasks are handled by a worker process.

The process model truly shines when it comes time to scale out. The share-nothing, horizontally partitionable nature of processes under the Twelve-Factor App methodology means that adding more concurrency is a simple and reliable operation, perfectly suited to on-demand cloud resources.

Configuration

In terms of configuration, or what the Twelve-Factor App calls config, the methodology dictates strict separation of config from code. In a cloud-native world, this makes sense: config varies substantially across deployments; code does not. The Twelve-Factor App model suggests that applications should read their config from the environment. The implication here is that, wherever you store your configuration information, it should be distinct from the application itself.

While this approach works well for many cloud-native applications, it's not hard to see how it would present challenges in virtualized operator networks, where SDN controllers may need to configure hundreds of downstream devices at runtime. In these environments, each "application" represents a decomposed process within a formerly discrete physical network device. When the environment encompasses thousands of networking devices from multiple vendors, having each decomposed process be responsible for getting its own config from the environment simply won't scale.

Rather, operators need a way to centralize the management of configs for all of the networking "applications" in the environment. Effectively, we need a way to "recompose" some of the functionality that was decomposed in making the application cloud-native. This is the only viable way that service orchestrators and SDN controllers can make sense of VNF microservices as being part of a discrete "network device." We could invest a huge amount of time and resources into inventing something new to accomplish this. Instead, it makes much more sense to reuse the same tools that operators use to centrally manage device configurations today: YANG data modeling and the NETCONF protocol.

8. Configuration Management for Cloud-Native VNFs

Let's take a closer look at what's involved in configuration management for cloud-native VNFs. According to the 2017 NCTA technical paper Cloud Native Network Function Virtualization: True Cloud for NFV, there are two separate domains to address: the control and management plane, and the data plane.

The control and management planes deal with configuration and session establishment. As these are workflow- and transaction-based systems, it is fairly straightforward to translate them into cloud-native applications. As for the data plane, it differs drastically from most cloud-native applications, as data planes are not transaction-based systems and can't rely on load-balancers to handle high-speed packets. Rather, SDN controllers and service orchestrators are used to direct the packet streams to the right data plane containers as part of normal routing and switching. In other words, load-balancing has been absorbed by networking.

Separating "Cattle" from "Pets"

To understand the implications of this issue, let's review a common analogy for scalability in the web-scale domain: "cattle" vs. "pets."

For microservices that can be designed to be stateless and easy to scale out (typically, microservices for the control and management plane functions of a VNF), developers should design and implement them to run as cattle. This implies that many identical

instances of the microservice will be running simultaneously, depending on the workload of the VNF application. These cattle, or stateless microservices, will be highly available, and can be taken down and replaced as necessary. They support service discovery. And distributed configuration stores such as etcd or Consul can be used for the automated discovery of configuration and topology information for individual microservices.

For microservices that need to be stateful, however, developers should treat these applications more like pets—unique, individual entities that require their own special care and feeding in order to scale. These applications must be run as a group of stateful microservices of a certain size, and one of these groups may have different resource requirements and/or runtime behavior than others. Here, configuration management is typically needed to define the different behaviors of these applications at deployment and/or runtime.

As described previously, from the point of view of network service orchestration, it doesn't make sense for the individual microservices within a VNF to be individually managed. Rather, operator environments require a special kind of microservice that serves configuration management for all of the different microservices that make up a given VNF application. This type of "configuration management microservice" is stateful, so should be implemented as a "pet." It will often be sufficient to run in an active/standby mode and doesn't need to scale out. In scenarios with less stringent uptime requirements, configuration management microservices can just be restarted on demand, without the need for standby instances.

9. Network Programmability for Cloud-Native VNFs

As discussed in a previous Tail-f white paper, "Trends in NFV Management," NEPs have to move away from the concept of "configuring" network appliances and begin to view their virtualized network functions as software to be programmed. As we evolve to a world where intelligent software drives the network, instead of human beings, we need to think of network management in terms of providing a programmable interface into network elements that this software can use.

As NEPs support and enable this network programmability paradigm shift, they begin to allow their network operator customers to fully automate their environments. This is an essential prerequisite to network agility, faster time-to-revenue, and DevOps ways of working. They also unleash innovation, as operators can now mix and match multivendor VNFs into new service chains and do things that even the vendors themselves never imagined.

Shifting to programmable interfaces for VNFs doesn't mean that network engineers now have to run out and get computer science degrees. Modern NFV tools can hide much of the lower-level detail involved in programming virtualized functions. But NEPs do have to start thinking about their VNFs as programmable software, rather than a device that an operator has to imperatively poke to perform some function. They need to think of configuration management in terms of a standards-based API into their devices.

10. SDN Moves to the Cloud

At the same time that NEPs are looking to reimagine their VNFs for cloud environments, a similar evolution is happening in the world of SDN. With the widespread adoption of SDN in data centers, we are seeing a shift towards SDN designed to run in cloud environments.

As discussed previously, in a software-defined network, the data comprising network traffic (data plane) is separated from the data that keeps the network running (control and management plane). Deployed as a virtual appliance in the network, an SDN controller handles southbound communications with the cloud-native VNFs and enables northbound communications between the applications running on the network and the controller itself. This capacity for real-time communication between applications and the network allows software-defined networks to do what can't be done with any purely physical network: reshape the data plane to suit the requirements of the applications.

Today, the SDN community has standardized on two software-based controllers:

- **ONOS**, a project, led by some of the creators of the OpenFlow protocol, aims to enable scalable network functions on telco infrastructure.

- **OpenDaylight** is now used predominantly by data centers that host network functions using OpenStack.

In addition, the Linux Foundation umbrella organization has unveiled the ONAP Project. This relative newcomer was formed to combine two previously separate open networking and orchestration projects: open-source ECOMP and Open Orchestration Project (OPEN-O). The goal of the ONAP Project is to develop a unified architecture and implementation, while supporting collaboration across the open-source community.

These SDN projects are all migrating towards cloud-native in their own ways.

ONOS
One of the projects under ONOS is called Central Office Re-architected as a Datacenter (CORD). The CORD architecture combines SDN, NFV, and elastic cloud services—all running on commodity hardware—to build cost-effective, agile networks. These networks should have significantly lower CAPEX/OPEX than conventional service provider architectures, and enable rapid service creation and monetization.

ONOS can now import device-specific NETCONF/YANG models, automatically manage these models in its distributed database, and dynamically sync and apply this configuration to the network.

OPNFV
Open-source group OPNFV has recently taken a big step forward in the cloud-native direction with its latest release, Euphrates. This fifth software release from OPNFV integrates Kubernetes and containers, as well as cross-community continuous integration capabilities. It delivers a pretested set of interoperable open-source NFV components that are key pieces of a commercially deployable NFV infrastructure.

Euphrates marks a major step towards the cloud-native capabilities that network operators say they want. It advances the cloud-native agenda of fast, scalable, vendor-neutral deployments in a number of ways. First, OPNFV has built architectural support around Kubernetes integration into the new release. They've also added support for upstream components such as FD.io and OpenDaylight, as well as the ability to do containerized OpenStack via Kolla.

ONAP

ONAP is an open-source platform that delivers capabilities for the design, creation, orchestration, monitoring, and lifecycle management of VNFs, SDNs, and higher-level services that combine them. ONAP provides for automatic policy-driven interaction of these functions and services in a dynamic, real-time cloud environment. ONAP has recently released an initial release of its production-ready source code and documentation to the open-source community in order to increase collaboration.

As for ONAP's migration to become cloud-native, the group recently conducted a demonstration at the Open Networking Summit. There, they demonstrated the integration of open networking and cloud-native technologies with Kubernetes, enabling ONAP to run on any public, private, or hybrid cloud network.

Network Programmability Fuels Cloud-Native SDN

Just because VNFs are moving towards being cloud native, that doesn't obviate the need for VNFs, either cloud-native or VM-based, to be programmable. Indeed, VNF programmability remains essential to allow the entire SDN network to be automated. And that, after all, is the ultimate goal of all of these projects: making the network more agile, accelerating time-to-revenue, and enabling DevOps way of working.

ONOS, OpenDaylight, and ONAP are all big proponents of model-driven methodology for defining the network configuration. For all of these projects, NETCONF is the protocol of choice as the southbound interface between SDN controllers/service orchestrators and the VNFs.

As described previously, when VNFs are operating in a cloud-native SDN environment, the NETCONF server component can be run as a microservice functioning as a configuration server in an active/standby mode. The job of this type of microservice is to take NETCONF configuration requests from the SDN controllers or network service orchestrators. It then pushes out the necessary changes, either directly or through the container orchestration software, such as Kubernetes, to the other stateless microservices of the VNF via their supported REST APIs or message bus.

11. Conclusion

As the industry moves towards running VNFs in cloud environments, NEPs face new challenges. Previously, they had to reimagine their devices as programmable virtualized software instances. Now, they must go a step farther: building those VNFs in a cloud-friendly way, so that their component microservices can take full advantage of on-demand cloud infrastructure and resources, continuous delivery, and DevOps.

In tackling this challenge, NEPs need to ensure that their VNFs are built with SDN requirements in mind, as SDN adoption is growing rapidly in the data center market and among their core customers. The best way to meet the needs of operators' SDN environments: build VNFs that are programmable.

By building in support for the industry-standard modeling language (YANG) and network configuration protocols (NETCONF, RESTCONF), NEPs will empower their service provider customers to fully automate their networks. And NEPs, service providers, and their customers will all benefit from the more agile, flexible, and effective applications that now become possible.

12. About Tail-f

Tail-f, a Cisco company, has been a leader in network programmability in the NFV space since its inception, and is a key contributor to the IETF, helping to define and develop new standards based on NETCONF and YANG. Our ConfD product can be deployed in a container and provides a simple, easy-to-implement way to add NETCONF and YANG capabilities to any vendors' VNFs, and help their customers fulfill the promise of fully programmable cloud-native virtual networks. Visit the Tail-f website for application notes on running ConfD on Docker, and on adding fully synchronous active-active high-availability cluster capabilities to ConfD.

To learn more, visit http://www.tail-f.com

tail-f a Cisco company

www.tail-f.com
info@tail-f.com

Corporate Headquarters

Sveavagen 25
111 34 Stockholm
Sweden
+46 8 21 37 40