Ericsson Review

The communications technology journal since 1924

2015 • 3

OpenStack as the API framework for NFV: the benefits, and the extensions needed

April 2, 2015

ERICSSON

OpenStack as the API framework for NFV: the benefits, and the extensions needed

Service providers are looking to Network Functions Virtualization (NFV) as a way to deliver and deploy Virtual Network Functions (VNF) services in a flexible way, using virtualization and cloud computing techniques. As an IaaS framework constructed as pluggable API components, OpenStack provides a given level of automation and orchestration to deploy and provision NFV services. But is it enough, and what improvements are needed?

ALAN KAVANAGH

Both OpenStack and NFV have developed considerably over the past few years from an IS/IT and a telecoms perspective. While these two concepts relate to similar areas – virtualization, REST-based APIs, and providing fast large-scale services independent of underlying hardware – they address these areas from different angles.

On the one hand, ETSI NFV aims to define an architecture and a set of interfaces so that physical network functions, like routers, firewalls, CDNs and telco applications, can be transformed: from software applications designed to run on specific dedicated hardware into decoupled applications – called VNFs – deployed on VMs or containers, on generic servers.

OpenStack, on the other hand, addresses service provisioning and virtualization by providing an open-source software service framework that is API-driven and pluggable, enabling public and private clouds to be quickly deployed and managed effectively. A high-level architecture of a telco cloud service built using OpenStack is shown in **Figure 1**.

The transformation to VNF services and deployment scenarios needs an API framework, and OpenStack is a suitable candidate. However, to ensure carrier-grade service and support for provisioning of NFV services, some extensions to the set of APIs are needed, and the concept as a whole needs to be embraced.

A modular architecture

In 2010, Rackspace and NASA jointly launched the first release of OpenStack distribution (Austin). Their aim was to create an open-source cloud platform that could provision computing, networking and storage services for private and public clouds. In other words, a large-scale and feature-rich API pluggable framework enabling automated service deployment and provisioning.

The original OpenStack architecture was modular, built with independent components (services), called through a REST-based API frontend. The initial Rackspace/NASA release included just two services: Nova – for managing compute resource pools; and Swift – the object storage system. The architecture is still modular today, but as **Figure 2** shows, it has grown with each release, through the addition of new components providing extra services in the IaaS layer.

As an IaaS framework that is flexible and API-driven, OpenStack offers vendors and solution providers a means to integrate their compute, network and storage-infrastructure plugins best suited to their environment. As such, OpenStack enables end-to-end service deployment, provisioning and orchestration, reducing implementation time from weeks to hours. The core services in OpenStack today include:

Horizon – a web-based service portal that provides tenants and administrators with a user interface for provisioning services such as VMs and object storage, and other capabilities like assigning IP addresses, and configuring access control for provisioned services.

Nova – which is responsible for compute services, such as scheduling, and on-demand initiation of VMs, Linux

BOX A Terms and abbreviations

AMQP	Advanced Message Queuing Protocol	KVM	kernel-based virtual machine
API	application programming interface	libvirt	virtualization API
BIOS	basic input/output system	NFV	Network Functions Virtualization
CDN	content delivery network	OVS	open virtual switch
CEE	Cloud Execution Environment	REST	Representational State Transfer
CLI	command-line interface	TaaS	tap as a service
CPU	central processing unit	TXT	Trusted Execution Technology
HOT	Heat Orchestration Template	VM	virtual machine
HTTPS	Hypertext Transfer Protocol Secure	VNF	Virtual Network Functions
IaaS	infrastructure as a service	vNIC	virtual network interface card

Containers (LXC), or Docker containers, as well as the removal of these services.

Neutron – which provides networking as a service to other OpenStack components (such as Nova). It does this by creating and attaching the virtual switch port to the vNIC of the VM, assigning the IP address, configuring network overlay for tenant isolation, and providing network configuration for baremetal-provisioned servers.

Swift – which provides multi-tenant object storage with inherent replication and automatic scaling. It manages large volumes of unstructured data, which is accessed through a RESTful API.

Cinder – which provides persistent block storage for instances, such as a VM, running on the OpenStack platform. It also manages block storage devices and volume snapshots.

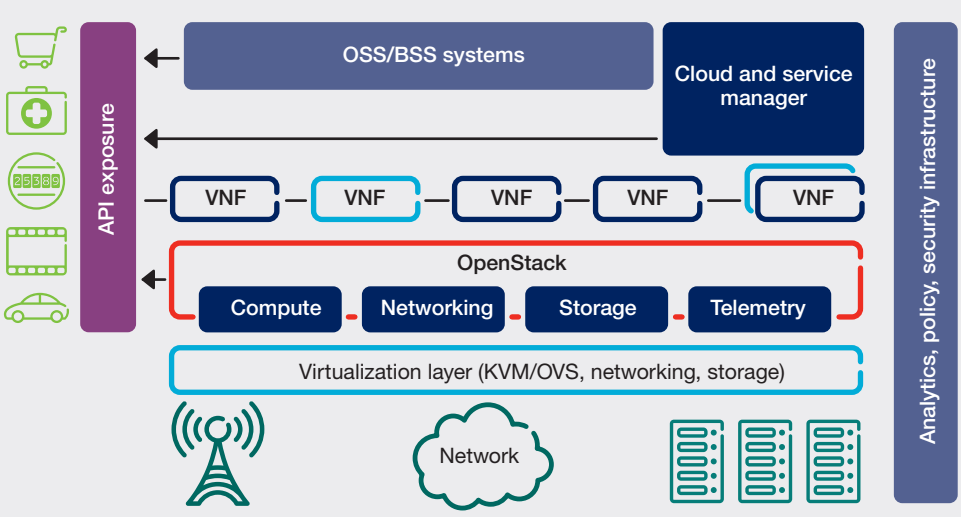
Keystone – which provides authentication and authorization for OpenStack services, tracking and authentication of users, as well as authorization of the services requested by a user (before the service request is processed). This is a common service used by all OpenStack API servers.

Glance – which stores and retrieves VM disk images and corresponding metadata.

Heat – which provides orchestration of services using Heat Orchestration Templates (HOTs), which describe a given cloud application and how it is deployed using OpenStack.

Ironic – which provisions baremetal machines using a PXE boot, for example. By image provisioning baremetal machines in an automated and orchestrated way, Ironic can provide high performing compute clusters, without incurring the overheads and license fees associated with hypervisors. This is a significant advantage for applications and services that perform high packet processing, and require deterministic performance as well as low latency.

FIGURE 1 **Telco cloud service architecture on OpenStack**



Functional blocks

The core of each OpenStack service, which is commonly referred to as the controller, manages the given service. Services like Nova and Neutron are built on a pluggable architecture and use an API web-based services frontend for managing the controller. The frontend is responsible for handling authentication and authorization of API calls via Keystone, as well as command and control functions like requesting or deleting a VM or establishing admin/user rights. As **Figure 3** shows, an OpenStack service calls another OpenStack service through its northbound API. Initial service requests are sent through a service portal, which can be the default dashboard (Horizon), or through a more enriched cloud manager that interfaces with the northbound API of the OpenStack controllers. Another method is to access each individual OpenStack service directly via the CLI, though this is most commonly used for troubleshooting and advanced administrator tasks.

For example, the Nova compute controller comprises a set of services including:

- Nova Scheduler – which determines where the compute service should be instantiated;
- Nova Conductor – which acts as a proxy for requests;
- Nova Compute Agent – which runs on the compute blade; and
- Nova database (db) – which stores most build time data (resource availability, consumption and state) for what instances are running and which compute blade they are running on.

The tasks these controller elements carry out to complete a service request are best described through the steps in the process to deploy a VM – a common task carried out by tenants logged on to a (Horizon) service portal.

VM boot process

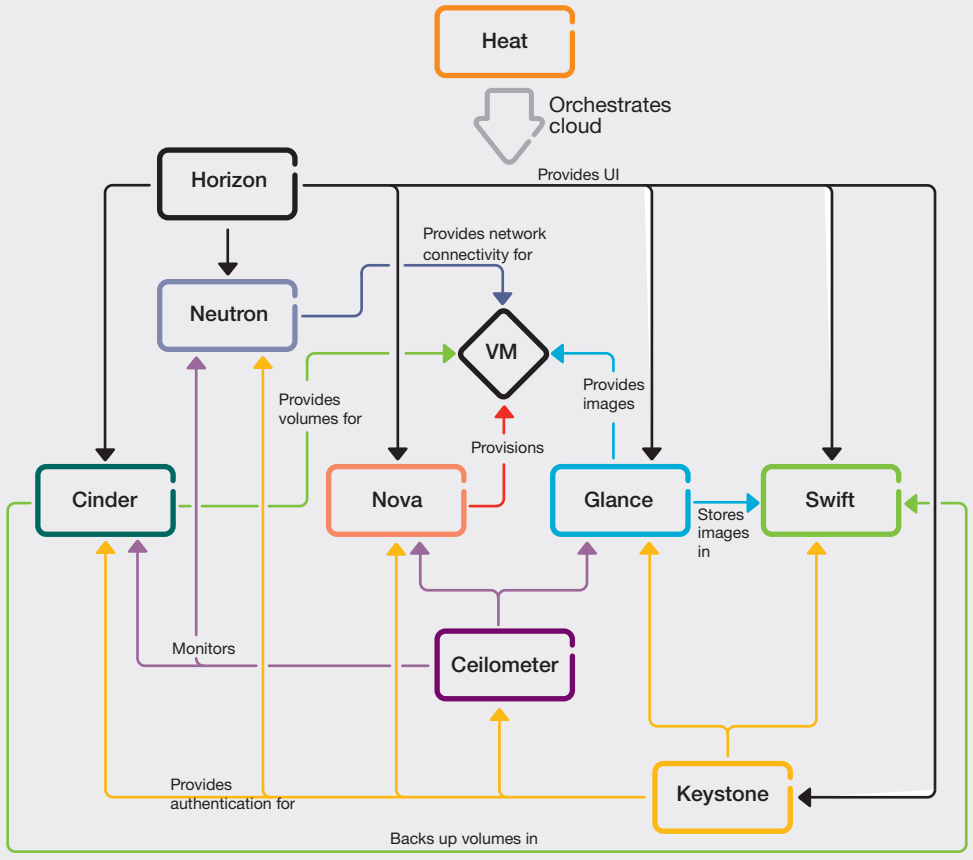
The API query for VM deployment is first authenticated by Keystone. If successful, the request is passed on to the Nova Scheduler, which allocates a compute blade for the VM, and then publishes the request, via the message queue, to the Nova Compute Agent.

The message queue is used for communication between all OpenStack daemons and uses AMQP – typically implemented with RabbitMQ.

Depending on the hypervisor or container solution being managed, the Nova Compute daemon will call the relevant plugin and the relevant API to instantiate a VM, for example, via the appropriate hypervisor. If the deployed solution is a KVM hypervisor, the Nova Compute Agent calls libvirt to instantiate a VM, and then updates the Nova db with the status of the requested VM.

Next, the Nova Compute Agent calls Neutron API to provision and configure the networking for the compute service. This may include attaching the VM

FIGURE 2 **OpenStack conceptual architecture**



to the network as well as allocation of an IP address.

After network provisioning, Nova Compute Agent calls Cinder API to provision persistent block storage based on tenant preferences.

Glance API is then called, and returns the URL denoting where the VM image file is stored in the backend object store, which Nova Compute Agent uses to download it. Once the image is installed on the compute blade, the VM will boot.

The steps in this process indicate how VNFs can be orchestrated, and automatically provisioned and deployed, using OpenStack services in a VM environment. Provisioning of VNFs via baremetal would also follow a similar process – which is supported by Ironic. VNFs that require all available resources on a given compute blade, such as RAM, CPU cores, disk I/O and/or full NIC bandwidth, are best suited to be provisioned on baremetal – without a hypervisor. In this case, the Nova Compute uses a baremetal plugin to call the Ironic API server that queries the Ironic Conductor to fetch the image files. Once Neutron has provisioned and configured the required networking, the baremetal node is deployed, and PXE boot is initiated to retrieve the VNF application until the node is rebooted and up and running with the VNF application.

For VNFs that are CPU heavy, memory intensive, and have high database transaction frequency, the Ironic API service is important. This is because it provides automated deployment and provisioning of the VNF or any application on baremetal servers, and removes the need for a hypervisor, which in turn reduces operating costs and complexity.

Pluggable architecture

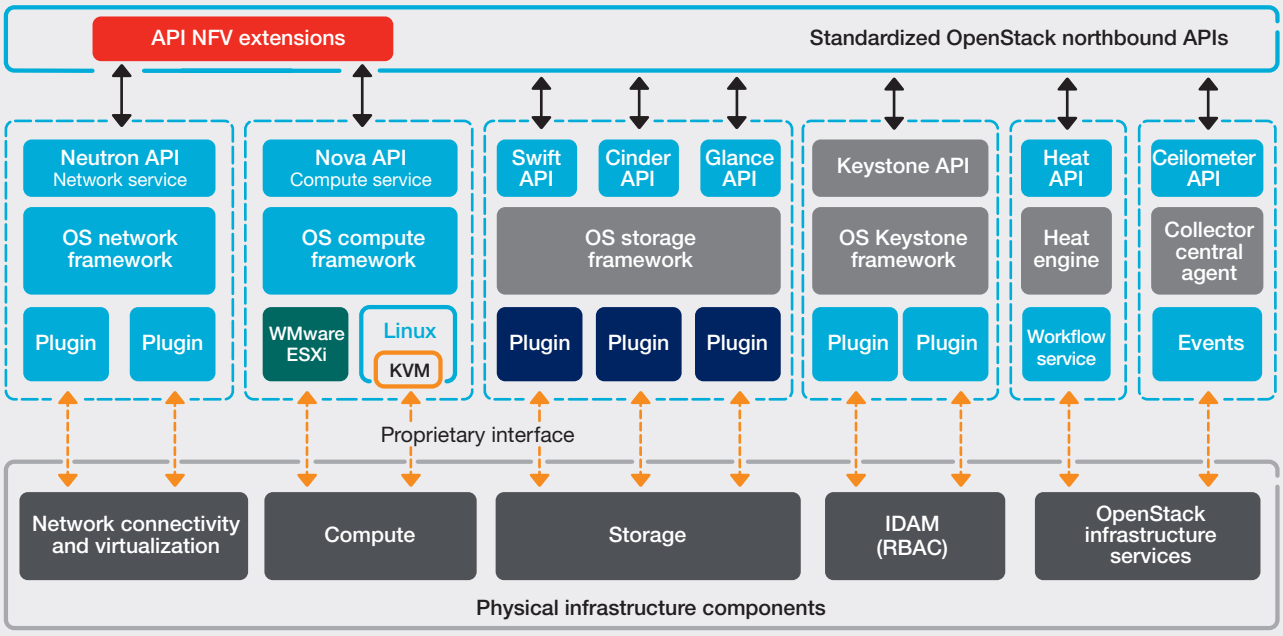
The main advantage of OpenStack is the pluggable nature of its framework architecture. As visualized in Figure 3, the flexibility offered by such an architecture allows service providers to choose the best backend solutions, which can be connected through the appropriate plugin. Which vendor plugin is most appropriate depends on the services the cloud provider wants to offer, the infrastructure being deployed, and the available vendor solutions.

Taking Neutron and Ericsson as an example, an Ericsson Layer-2/Layer-3 solution would use the Ericsson Neutron plugin. The pluggable architecture offers OpenStack Neutron a way to extend its functionality with more advanced networking solutions, such as firewall, VPN, load balancing and port mirroring, implemented as standalone service plugins. In this way, networking modules in Neutron can provide additional and much needed functionality that can be selected and included in the overall solution based on the requirements of the service provider.

The Nova-Docker plugin has been recently developed to support the deployment of Docker containers via Nova Controller. While some VNFs take advantage of containers, they are not a complete replacement solution for VMs or baremetal deployment, but provide another deployment option that is suitable for some applications. In fact, as more lightweight container solutions – like Ubuntu's LXD – become available, the pluggable architecture of OpenStack really comes into play. Deployment and provisioning of the given container solution can be achieved by simply adding the specific Nova plugin and calling the OpenStack API to provision the VNF with a specific deployment option.

The pluggable architecture is ideal for VNFs that require specific networking configuration, such as VLAN trunking, or advanced network services like Layer-3 routing, as they require dynamic routing protocols to be provisioned in addition to multiple virtual routers. However, support or full implementation for such advanced services is not yet included in OpenStack, illustrating some of the gaps that remain to be fulfilled by Neutron to support all NFV deployment and configuration scenarios.

FIGURE 3 **OpenStack functional architecture**



What's needed?

OpenStack provides an IaaS on-demand cloud resource deployment and configuration service that enables VNFs to be deployed quickly (within a matter of minutes) on generic hardware through automatic deployment and provisioning of VNFs in a cloud environment. The benefits for NFV vendors and service providers include:

- faster time to market – reducing the typical lead time from weeks to minutes;
- elastic scaling of VNF services – which results in maximum utilization of hardware resources and reduces capex and opex;
- support for various compute resources and flavors – offering several deployment options such as baremetal, containers, and hardware virtualization;
- automated continuous deployment and rolling upgrades; and
- pluggable backends – allowing vendors and service providers to provide innovative solutions based on deployment and service needs.

As an open-source project, OpenStack is licensed under Apache 2.0 and, as such, provides a basis to develop the necessary plugins to support the provisioning and deployment of a large number of VNFs, and to develop extensions to API services, that are needed to support vendor and service provider VNFs.

Future challenges

When OpenStack began in 2010, the target market and predominant scenarios addressed by the OpenStack community focused on traditional three-tier web services and web content hosting. Such services are typical for public clouds, as they are well suited for deployment on endpoint IP servers with a small db using a hypervisor-type solution. Over time, the focus of OpenStack has evolved to include support for:

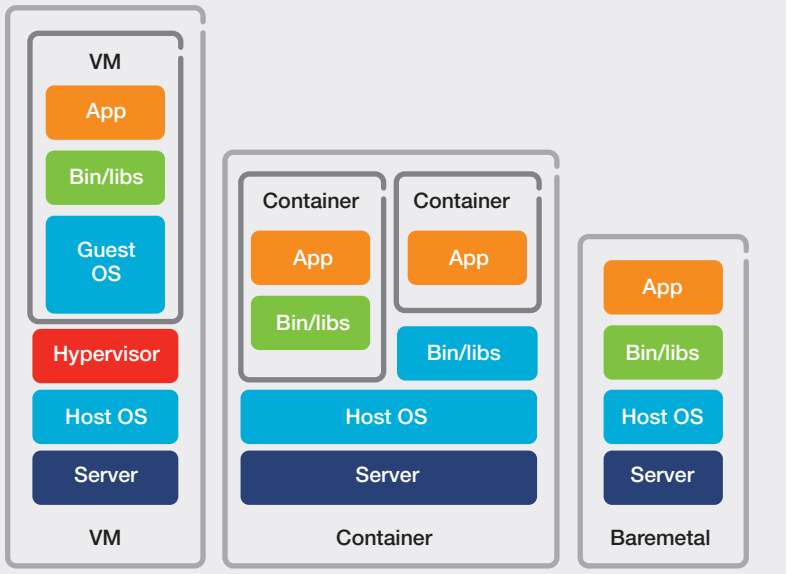
- telemetry services – Ceilometer;
- an orchestration engine for infrastructure life cycle management – Heat;
- big data – through the Sahara project, which manages Hadoop clusters; and
- advanced services in Neutron – such as remote VPN access support, load balancing and firewall as a service.

In 2013, Ericsson and AT&T started to drive and include support for NFV through several key contributions, such as VLAN aware VMs, VLAN trunking, dynamic logging, soft-affinity policy for server groups in Nova and multi-vNIC per VM. In addition, Ericsson experts are core team members in the Ceilometer and Barbican projects, as well as a number of lead contributors in projects such as the Telco Working Group and Nova in OpenStack.

However, a number of key features are still needed to enable NFV to embrace OpenStack and the infrastructure components it manages. This is where the challenge lies ahead, together with providing assurance that OpenStack does not result in service degradation, and that it supports the necessary configuration options to deploy VNF services from different vendors. For the moment, further development of some critical features is required before NFV services can be fully deployed and provisioned to run reliably in a predictable and deterministic manner.

One of the key features currently under development relates to how

FIGURE 4 **VNF deployment options**



Nova determines where to place an application – specifically, the use case where a VNF requires a cluster of machines, with individual applications placed on different compute blades. Such a setup can be achieved by setting the ServerGroupAntiAffinityFilter policy. However, the anti-affinity concept cannot be extended to network and storage backends to address cases where the VNF spans several storage pods. Currently, Nova decides where a given VNF application should be placed based on simple weights and filters; it does not take into consideration, for example, that a VNF should run on a specific blade with a dedicated storage drive.

A critical feature missing from Neutron, for example, is support for VLAN trunking, which is needed to deploy and configure VNF services, where the VNF has selected its own VLAN ID – a typical feature for infrastructure services – instead of using the one assigned by Neutron. Support for VLAN trunking is available in some vendor solutions like the Ericsson OpenStack CEE distribution, making the most of the flexibility of OpenStack to include vendor additions.

Work is ongoing to provide Layer-3 services in Neutron. This work is in the form of providing support for provisioning and configuring dynamic-routing protocols, such as OSPF and BGP – which are used for load balancing, dynamic route announcement and route distribution among VNFs in a cluster – and MPLS/BGP VPN – which is used for inter data center VPN connectivity.

Yet another missing VNF service is the capability to request provisioning and configuration for virtual routers in tenant networks. While IPv6 support has been added in the last two release cycles – Icehouse and Juno – feature parity with IPv4 still requires the addition of a number of features. Similarly, a number of capabilities are missing, including:

- the ability to set QoS per VNF service per tenant;
- fast detection and recovery of network faults at the overlay layer;
- port mirroring, which is used for purposes such as troubleshooting; and
- auditing and traceability services at the different layers in the cloud stack.

Within the OpenStack community, Ericsson is working on an implementation for port mirroring under the TaaS project – contributing the source code for the TaaS service.

Traceability tools are vital for locating service failures at various points in large systems and for recovering crashed services – particularly for virtualization open source service components like KVM and OVS that are managed and interfaced by several different OpenStack components. Ericsson and other industry players have added health checks and watchdogs to OpenStack components (like libvirt) and layers (like KVM) outside the OpenStack system, some pre-runtime and runtime checks still need to be covered. For example, VNFs need to have a no service interruption guarantee to enable carrier grade services and fast failover detection.

To authenticate API calls, OpenStack uses rule-based access control and role-based access control for authenticating user and tenant access within the defined set of configured roles. Yet, a number of threats remain, both in OpenStack and in the underlay system – which OpenStack does not control or manage. In the Juno release, support for encrypting metadata traffic, via HTTPS, was included. Some services such as boot attestation, host attestation and firmware validation, tend to be performed outside the management scope of OpenStack. These services use, for example, Intel's Trusted Execution Technology (TXT) to ensure their trustability by storing hash values on an attestation server. These values are then queried, for example, after a baremetal blade has been returned to the Ironic pool before being made available to other tenants, to attest and verify that hardware and software BIOS and firmware have not been tampered with. This is an essential feature for VNFs to ensure they are provisioned and run in a secure environment. The ability to validate that physical resources continue to operate as expected is yet another essential feature. OpenStack lacks the capability to check that resources are still functional before providing them to a tenant. For example, the ability to check that SSD drives are still functional on assigned blades before the Ironic service allocates them to a tenant is missing.

As an essential component for guaranteeing service assurance, SLAs enable VNFs to run in a predictable environment. As such, SLAs are an essential element of assuring that VNFs run in a deterministic environment, so that

a dedicated number of CPU cores (CPU core pinning) can be assigned to VNFs, memory allocation is guaranteed, and sufficient page table sizes are reserved among other features one would require for providing carrier-grade functional control for VNF provisioning.

Conclusion

OpenStack is still evolving and will take time to mature. New features and extensions designed to address the specific requirements of NFV in telco and enterprise use cases will enable NFV services to be provisioned and deployed. This will, however, require a number of future releases. While some development work still needs to be carried out by the OpenStack community to support all the necessary features and ensure that OpenStack is carrier grade, its extendable and pluggable services framework provides vendors and service providers with a flexible solution for interfacing a multitude of plugins and backend solutions.

A large number of solutions may be developed to suit service provider needs, and so a certification program is required to ensure which plugins and infrastructure blocks are connected and NFV certified.

Alan Kavanagh

is a cloud system architect expert working in Development Unit Networks & Cloud, Systems and Technology. He has over 15 years' experience in fixed and mobile broadband networks in the areas of standardization, system design and R&D. Over the past number of years he has been working on designing and building innovative solutions related to cloud computing in the areas of OpenStack, NFV and PaaS. He holds a B.A. in computer and electronic engineering, and a B.A.I. in mathematics from Trinity College Dublin (TCD), Ireland.

Additional information

OpenStack community and projects: http://www.openstack.org

Ericsson Review

To bring you the best of Ericsson's research world, our employees have been writing articles for Ericsson Review – our communications technology journal – since 1924. Today, Ericsson Review articles have a two-to five-year perspective and our objective is to provide you with up-to-date insights on how things are shaping up for the Networked Society.

Address:
Ericsson
SE-164 83 Stockholm, Sweden
Phone: +46 8 719 00 00

Publishing:
Additional Ericsson Review material and articles are published on: www.ericsson.com/review. Use the RSS feed to stay informed of the latest updates.

Ericsson Technology Insights

All Ericsson Review articles are available on the Ericsson Technology Insights app available for Android and iOS devices. The link for your device is on the Ericsson Review website:www.ericsson.com/review. If you are viewing this digitally, you can:
download from Google Play or
download from the App Store

Publisher: Ulf Ewaldsson

Editorial board:
Joakim Cerwall, Stefan Dahlfort, Åsa Degermark, Deirdre P. Doyle, Björn Ekelund, Dan Fahrman, Anita Frisell, Jonas Högberg, Geoff Hollingworth, Patrick Jestin, Cenk Kirbas, Sara Kullman, Börje Lundwall, Hans Mickelsson, Ulf Olsson, Patrik Regårdh, Patrik Roséen, Gunnar Thrysin, and Tonny Uhlin.

Editor:
Deirdre P. Doyle
deirdre.doyle@jgcommunication.se

Subeditor:
Ian Nicholson

Art director and layout:
Carola Pilarz

Illustrations:
Claes-Göran Andersson

ISSN: 0014-0171

Volume: 92, 2015

Ericsson
SE-164 83 Stockholm, Sweden
Phone: + 46 10 719 0000

ISSN 0014-0171
284 23-3254 | Uen