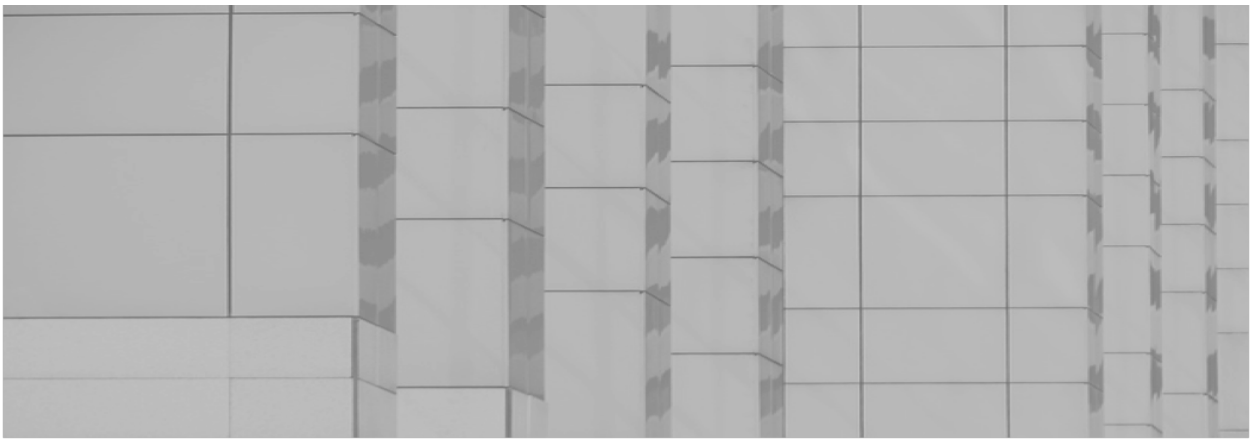
WHITE PAPER



SDN/NFV

XOS: Service Orchestration for CORD

This white paper describes XOS, a service orchestration layer built on top of OpenStack and ONOS that manages scalable services running in a Central Office Re-architected as a Datacenter (CORD). XOS makes it possible to create, name, operationalize, manage and compose services as first-class operations, and in doing so, unifies the management of SDN-based control applications, NFV-based data plane functions, and traditional cloud services.

June 1, 2015

Introduction

This paper describes XOS, an orchestration layer that manages a collection of services running in a Telco Central Office Re-architected as a Datacenter (CORD). XOS defines a coherent framework for combining SDN, NFV, and Cloud services, all running on commodity hardware, to build cost-effective and agile networks.

XOS builds on two open source projects. The first is OpenStack, which manages virtual resources on a cluster of commodity servers. OpenStack is responsible for creating and provisioning virtual machines (VMs) and virtual networks (VNs), while XOS defines a service abstraction on top of those virtual resources. The second is ONOS, which manages the cluster's switching fabric. ONOS hosts a collection of network control applications, while XOS incorporates those applications into the overall portfolio of CORD services.

XOS provides explicit support for multi-tenant services, making it possible to create, name, operationalize, manage and compose services as first-class operations. Commodity multi-tenant clouds are designed to host applications, but they usually treat these as single-tenant services that run on top of the cloud, for the benefit of the user. In contrast, XOS provides a framework for implementing multi-tenant services that become part of CORD, thereby lowering the barrier for services to build on each other.

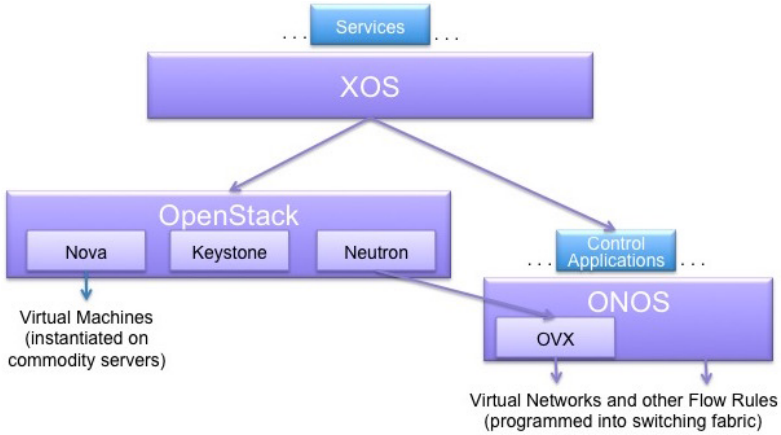


Figure 1. Relationship between XOS, OpenStack, and ONOS.

Figure 1 gives a high-level depiction of XOS, OpenStack, and ONOS, including their critical subsystems. As shown in the figure, XOS supports a set of services and ONOS hosts a set of control applications. OpenStack's Neutron subsystem uses ONOS's OVX subsystem to embed virtual networks in the underlying fabric. (In effect, Neutron can be viewed as one of the control apps running on ONOS.)

Unifying Framework

One of the main values that XOS brings to CORD is a set of abstractions that unify three related but distinct threads:

- The first is SDN, which is about separating the network's control and data planes. This makes the control plane open and programmable, leading to increased innovation. It also allows for simplification of forwarding devices that can be built using merchant silicon, resulting in less expensive white-box switches.
- The second is NFV, which is about moving the data plane from hardware appliances to virtual machines. This reduces CAPEX costs (through server consolidation and replacing high-margin devices with commodity hardware) and OPEX costs (through software-based orchestration). It also has the potential to improve operator agility and increases the opportunity for innovation.
- The third is the Cloud, which defines the state-of-the-art in building scalable services—leveraging software-based solutions, virtualized commodity platforms, elastic scaling, and service composition, to enable network operators to rapidly innovate.

To understand how XOS accomplishes this, it is necessary to start with its approach. We call XOS an operating system because it plays much the same role in CORD as a traditional operating system does on a computer: it provides general programming abstractions to support a wide range of applications, while at the same time safely multiplexing the underlying hardware resources and software services between them. In this XOS-centric view, OpenStack and ONOS play the role of two essential OS subsystems.

An OS provides many inter-related mechanisms to empower users. If we define an OS by a successful example, Unix, then an OS provides isolated resource containers in which programs run (e.g., processes); mechanisms for programs to communicate with each other (e.g., pipes); conventions about how programs are named (e.g., /usr/bin), configured (e.g., /etc), and started (e.g., init); a mechanism to program new functionality through the composition of existing programs (e.g., shell); a means to identify and authorize principals (e.g., users); and a means to incorporate new hardware into the system (e.g., device drivers).

XOS provides counterparts to all these mechanisms—as described in the next section—but in general terms, XOS adopts much the same design philosophy as Unix. Both are organized around a single cohesive idea—everything is a file in Unix and Everything-as-a-Service (XaaS) in XOS. Both also aim to have a minimal core (kernel)

and are easily extended to include new functionality. In Unix, the set of extensions correspond to the applications running top of the Unix kernel. This is also the case with XOS, where the core is minimal and the interesting functionality is provided by a collection of services.

Implicit in the XOS architecture is an underlying model of exactly what constitutes a service. Our model is that every service incorporated into XOS (1) provides a *service controller* that exports a programmable interface to network-wide functionality; (2) elastically scales across a set of *service instances*; and (3) is multi-tenant with an associated *tenant abstraction*. In CORD, the service instances correspond to OpenStack-managed VMs and ONOS-managed white-box switches.

Abstractions & Mechanisms

The main value XOS brings to CORD is a collection of OS abstractions and mechanisms that support services and service composition. Using Unix as a model, we summarize the abstractions and mechanisms as follows:

- Resource Containers: XOS provides a *Slice* abstraction (Unix process) in which *Services* (application programs) run, along with low-level mechanisms (e.g., fork/exec, /etc, and /init) for securely booting and managing a service in a slice.
- Application Programs: XOS defines a *Service* abstraction (application programs), along with a *Tenancy* abstraction that provides a means by which Services can be composed to create new functionality (Unix pipes).
- Programming Environment: XOS provides a *View* abstraction (Unix shell), a RESTful API (kernel interface), and a convenient user-level library called *xoslib* (libc), all of which can be used to extend XOS with new capabilities.
- Importing Resources: XOS defines a *Controller* abstraction that can be used to import new resources into XOS (device drivers).

Using Unix as a model for XOS is not perfect, but the mapping is helpful in understanding the roles played by the various XOS components, as well as to highlight the gap between creating resource containers and providing a full-featured OS. The rest of this section looks at these abstractions and mechanisms in more detail, with a focus on the role they play in supporting services in CORD.

Services and Slices

Services are a first-class abstraction (object) in XOS, meaning they are explicitly created, named, and managed. A Service is defined in terms a set of Slices, each of which hosts the scalable instances that implement the service, and a Controller that

represents the XOS end-point for the service's controller. (XOS supports both legacy service controllers and “native controllers” built entirely within XOS.)

A Slice, in turn, is defined to be a set of Virtual Machines (VM), each of which is instantiated on some physical server, and a set of Virtual Networks (VN), each of which is embedded in the underlying physical network and interconnects the VMs. XOS gives users control over the operational parameters of their slices (e.g., VM placement, VN topology, resource allocation), as well as provides a collection of low-level mechanisms to securely boot and configure a service in a slice.

Collectively, VMs and VNs isolate services from each other, where XOS is indirectly responsible for allocating resources to slices. We say indirectly because XOS uses OpenStack's Nova service to directly implement VMs and ONOS's OVX (invoked via OpenStack's Neutron interface) to directly implement VNs. The value of explicitly representing this functionality in XOS is that it provides a single uniform programming environment for operating on both slices and services.

Services and Service Composition

The power of building a system from modular components is realized when the system provides a means to compose those components. XOS provides explicit support for composing services, or to be more specific, for service providers to declare that “Service A is a tenant of Service B.” We represent this relationship in XOS as a *Tenant* object, which (1) binds a tenant service to a provider service, (2) specifies how the two services interconnect in the data plane, and (3) manages the credentials needed to connect the two services in the control plane.

Multi-tenancy is a staple of cloud services, but there are two assumptions that do not apply in the general case that XOS addresses. The first is that the tenant is a user (e.g., John Smith is a tenant of Amazon's EC2 service). Tenancy raises additional challenges when the tenant is itself an elastically scalable service; all the service instances must collectively be able to access the provider service.

The second is that services run *on* some cloud (e.g., EC2), but XOS is also designed to support services that are *part of* a cloud. This corresponds to the ecosystem of services offered *by* a cloud provider (e.g., the AWS portfolio, as opposed to tenant services that run on EC2). These services build upon each other and are operated by a single cloud provider, Amazon in the case of AWS and the Telco in the case of CORD.

XOS provides mechanisms that lower the operational costs of Service A being a tenant of Service B under these more general circumstances. We broadly characterize them as enabling composition in the data plane and enabling composition in the control plane.

- Data plane composition is about specifying how one service connects to (exchanges packets with) another service. Isolation is the default, with each service running on its own Slice. XOS provides a means to interconnect a pair of services by interconnecting their respective VNs.
- Control plane composition is about managing the tenancy of one service relative to another. XOS provides mechanisms that ensures each instance of service A has the credentials it needs to access service B.

Programming Environment

XOS provides a programmable environment, called a *View*, for customizing access to (and management of) the set of XOS services. Views are typically tailored for a particular user community or workflow. For example, XOS includes a traditional cloud tenant view (it mimics the EC2 interface); a service developer view (it exposes low-level control over VM placement, VN topology, and service composition); a service operator view (it is used to manage an already developed and deployed service); and a cloud operator view (it is used to define global policies and configuration parameters for a suite of services).

Views define a portal through which users (or their agents) access and control one or more scalable services, but a view is not itself a scalable service. For any function that requires scalable computation or state, the XOS design philosophy is to (1) encapsulate the scalable aspects of the function in a service, with a corresponding service controller and scalable set of service instances; (2) import the service into XOS via a Controller; and (3) run only the interface to that service in a view, accessing the service controller indirectly through *xoslib*.

Importing Resources

The abstractions described to this point are about defining an environment for programmers building services in XOS. Like any operating system, XOS must also manage a set of underlying hardware resources. In our case, this means importing Infrastructure-as-a-Service (IaaS) into XOS. Fortunately, IaaS-based services like OpenStack export a controller that can be imported into XOS.

CORD focuses on a single Central Office—and hence, a single OpenStack cluster—but XOS is general enough to support services (and slices) that span multiple Central Offices, as would be the case for a Telco-wide service that includes a point-of-presence multiple central offices.

CORD Services

XOS is deployed on different infrastructure with different service portfolios, including CORD. As described in more detail in a companion white paper, CORD supports a collection of scalable services created by virtualizing legacy hardware devices currently deployed in Central Offices: Optical Line Termination (OLT), Customer Premises Equipment (CPE), and Broadband Network Gateway (BNG). The end result is a collection of three elastically scalable, multi-tenant services:

- Access-as-a-Service (ACCaaS): Implemented by a control application (called vOLT) running on ONOS, where each tenant corresponds to a Subscriber VLAN.
- Subscriber-as-a-Service (SUBaaS): Implemented by a data plane function (called vCPE) scaled across a set of OpenStack-hosted VMs, where each tenant corresponds to a Subscriber Bundle.
- Internet-as-a-Service (INTaaS): Implemented by a control application (called vBNG) running on ONOS, where each tenant corresponds to a Routable Subnet.

CORD's service portfolio also includes a Content Distribution Network (CDN)—itself a scalable cloud service deployed throughout the operator's network, including caches in the central offices. This results in the tenancy graph shown in Figure 2, where the tenancy relationship is how service composition is specified in XOS (i.e., how VNs are interconnected in the data plane and credentials distributed in the control plane).

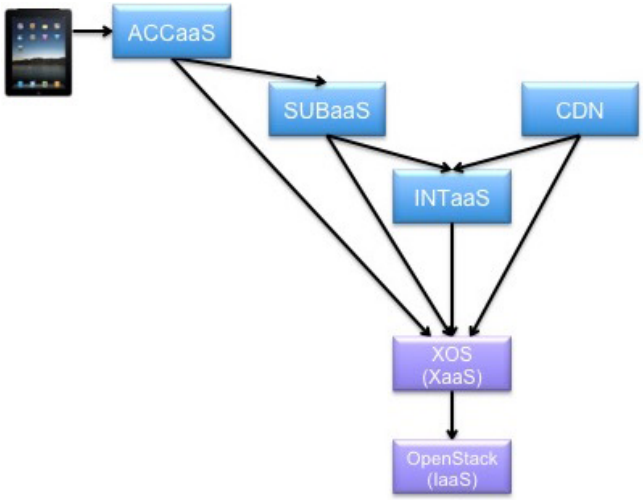


Figure 2. CORD Tenancy Graph.

In this particular configuration, the home device is a tenant of ACCaaS (connecting the device to the subscriber VLAN); ACCaaS is a tenant of SUBaaS (connecting the subscriber VLAN to a container that implements a subscriber bundle); both SUBaaS

and CDN are tenants of INTaaS (which provides a means for these services to reach the Internet); all four of these services are tenants of XOS (which represents these services as first-class objects); and XOS is, in turn, a tenant of OpenStack (which provides building block VMs and VNs).

Returning to the overall software architecture introduced in Figure 1, these four services all run in virtualized infrastructure managed by OpenStack; Access-as-a-Service and Internet-as-a-Service correspond to the vOLT and vBNG control applications, respectively, running on top of ONOS; and XOS manages the entire set of services and the relationships among them. This results in the specific software configuration shown in Figure 3.

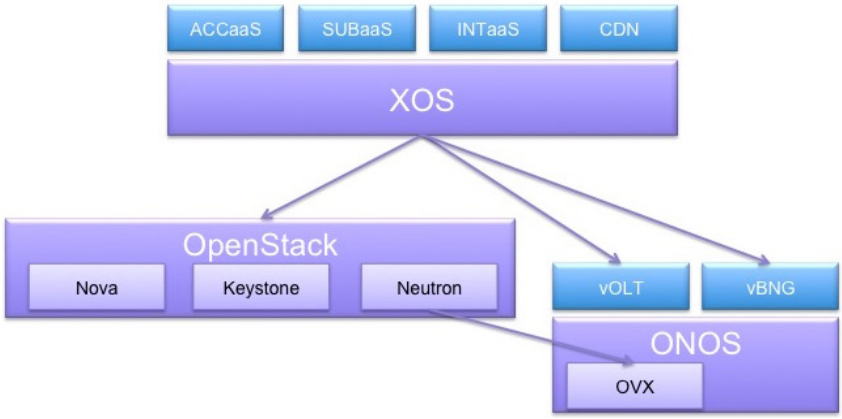


Figure 3. Software configuration revisited.

There are three important takeaways from this example.

- XOS (and the underlying Everything-as-a-Service) is powerful enough to be applied to an environment that has historically been implemented and operated at the device level. It is possible transition from device-focused to service-focused, making it possible to bring the agility of the cloud to bear on the Telco Central Office.
- The combination of services span the spectrum of a traditional cloud service (CDN), a typical NFV-based service (Subscriber-as-a-Service), and two SDN-based services (Access-as-a-Service, Internet-as-a-Service), all brought under the control of a single service orchestration layer. It is possible to accommodate both SDN and NFV based services under a common framework.
- Being able to isolate services on private virtual networks is an important part of the central office's security architecture. In this case, the VMs that implement the ACCaaS and SUBaaS services are connected by one VN that is not publicly

routable, while the SUBaaS and CDN services are connected by a second VN that is publicly routable. INTaaS is the scalable service that implements routing for this second VN. It is essentially a logical router—implemented as a control application running on ONOS, that in turn installs flow rules in the underlying switching fabric—connecting the other services running inside the datacenter to the rest of the Internet.

NFV Architecture

As highlighted throughout this report, XOS unifies the Cloud, SDN, and NFV under a single service-centric framework. This raises the issue of how CORD relates to the ETSI NFV Architecture, which is focused on moving functionality from hardware devices to VMs, but does not include a comprehensive story for SDN or the Cloud. We believe the CORD architecture can be incorporated into (mapped onto) the NFV architecture in a straightforward way, with XOS playing a central role in that integration.

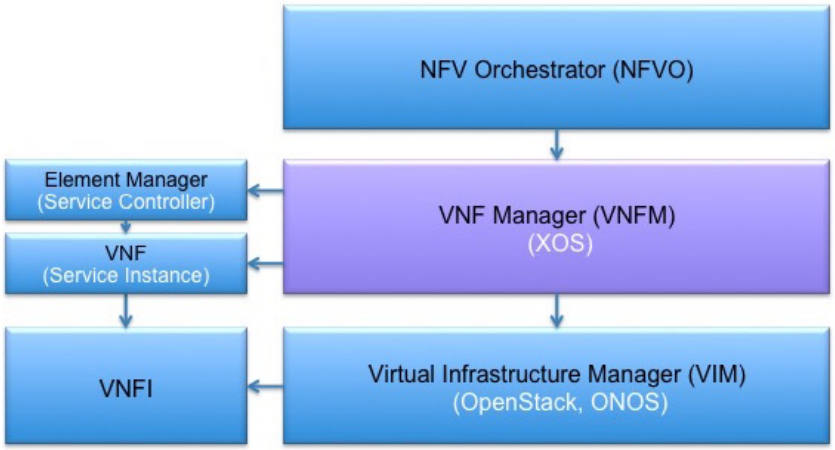


Figure 4. CORD Software Architecture Mapped onto the NFV Architecture.

Figure 4 gives a simplified depiction of the ETSI NFV architecture, annotated with the three key software components of CORD (XOS, OpenStack, and ONOS), plus two of the central abstractions of XOS (Service Controllers and Service Instances). That the Virtual Infrastructure Manager (VIM) corresponds to OpenStack and ONOS is no surprise. This matches the current practice, although ONOS plays a dual role: it both creates virtual networks on the underlying switching fabric and it hosts SDN-based services (e.g., ACCaaS and INTaaS). These services are not shown in Figure 4, but their service controllers (vOLT and vBNG) would be example Element Managers (EM).

With respect to the VNFs, there are two cases. First, SUBaaS and CDN correspond to the vCPE and Cache software running in VMs, similar to existing VNFs being demonstrated in OPNFV. Second, ACCaaS and INTaaS correspond to SDN programs

(flow rules) installed in the white-box switches. The controllers run in VMs, but the service functionality is implemented by the white-box switches. In other words, not all functionality need be implemented in VMs—CORD includes a combination of "program-based VNFs" running in VMs and "OpenFlow-based VNFs" running in the switching fabric!

Because XOS coordinates a collection of service controllers, it plays the role of a VNF Manager (VNFM). Interestingly, because XOS treats *everything* as a service—including the underlying IaaS—having the VNFM control both a set of EMs and the VIM is one of the biggest advantages XOS brings to the NFV architecture: It defines a single locus of control for coordinating both the infrastructure that hosts a VNF and the controller that manages a VNF. Our experience is that these two require careful synchronization.

Using XOS as the VNFM has a second advantage, which is to facilitate unbundling the NFV Orchestrator (NFVO). In particular, XOS Views provide an opportunity to build minimal/targeted orchestrators. Any given provider can program whatever policy-level functionality it needs into a customized View—including incorporating elements of their legacy OSS/BSS machinery—without having to adopt a one-size-fits-all third-party NFVO. Of course, it is also possible to implement an existing NFVO like HP's NFV Director or Telefonica's OpenMano on top of *xoslib*.

Summary

XOS is a service orchestration layer that runs on top of OpenStack and ONOS to manage scalable services running in a Central Office Re-architected as a Datacenter. XOS provides explicit support for multi-tenant services, making it possible to create, name, operationalize, manage and compose services as first-class operations. In doing so, XOS unifies management of a collection of services that are traditionally characterized as being NFV, SDN, or Cloud-specific. With XOS, all services—whether they run in a scalable number of VMs in the CO, as flow runs programmed into the CO switching fabric, or as part of a global cloud service—are managed as part of the operator's portfolio of scalable, multi-tenant services, with service tenancy defining the policy for how that portfolio is interconnected.

An important observation about the design is that XOS plays the role of VNFM in ETSI's NFV Architecture. Doing so has three advantages: (1) it empowers both the operator and the software service vender by unbundling the NFVO; (2) it brings both SDN and NFV under the same orchestration framework; and (3) it provides a convenient implementation point for addressing the pragmatic issue of simultaneously controlling and programing VNFs.

About ONOS

ONOS is the open source SDN network operating system for service provider and mission critical networks, architected to provide a resilient, high performance SDN control plane featuring rich northbound/southbound abstractions and interfaces to support a diversity of management, control, service applications and network devices. ONOS(Avocet release) was open sourced on December 5th, 2014. Blackbird, the second ONOS release in Feb 2015, demonstrated SDN control plane performance, scale and HA leadership.

ONOS ecosystem comprises ON.Lab, organizations who are funding and contributing to the ONOS initiative including Tier 1 Service Providers - AT&T, NTT Communications, SK Telecom, and leading vendors including Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, NEC; members who are collaborating and contributing to ONOS include ONF, Infoblox, SRI, Internet2, Happiest Minds, CNIT, Criterion networks, Black Duck, Create-Net, KISTI, KAIST, NAIM, Kreonet, and the broader ONOS community. Learn how you can get involved with ONOS at onosproject.org.