Cisco ACI Unified Plug-in for OpenStack Architectural Overview

First Published: February 2019

Table of Contents

Introduction 4

OpenStack and Neutron Overview 4
- Neutron Architecture 4
- Neutron Network Model 6
- Neutron and Security 6
- Challenges with OpenStack Neutron 6

Cisco ACI Overview 7
- Cisco APIC 7
- Cisco Nexus 9000 Series Switches 8
- Cisco ACI Software and Policy Model 10
- More Information About Cisco ACI 12

OpenStack and Cisco ACI 13
- Cisco ACI Unified Plug-in for OpenStack 14
- ML2 and GBP Networking Models 14
  - ML2 Networking Mapping to Cisco ACI 14
  - GBP Networking Mapping to Cisco ACI 15
  - The apic_aim ML2 Plug-in 18
  - Cisco ACI Integration Module (AIM) 18
  - OpFlex Proxy and OpFlex and OVS Agents 24
- OpFlex and PhysDom Deployments 25
  - OpFlex Node Deployment 27
  - PhysDom Node Deployment 29
- Optimized Routing, DHCP and Metadata Proxy Operations 31
  - Distributed Routing Function 31
  - Neutron DHCP Optimization Service 31
  - Neutron MetaData Optimization Service 32
- Support for Neutron Networks 33
  - External Neutron Networks 34
  - Dedicated External Network 39
  - Shared External Network 39

Infrastructure Architecture 43
- Cisco ACI and OpenStack Physical Architecture 43

Life of a Packet with Open vSwitch and OpFlex ACI Plug-in 45
- The Endpoint File 46
- Traffic Between Instances on the Same Host 49
- Traffic Between Instances on Different Hosts 49

Traffic Between an OpenStack Instance and an External Subnet 51
Appendix 53
OpenStack plug-in Constructs 53

Introduction

OpenStack provides an open-source framework for running infrastructure to support private, public and telco clouds. OpenStack is built as a modular architecture, implemented from various projects, that enables users to choose how to best implement compute, storage, network, and many other aspects of the solution. The OpenStack Neutron project is responsible for OpenStack networking.

Cisco Application Centric Infrastructure (ACI) is a data center software-defined networking (SDN) solution that provides centralized, policy-based fabric management and integrated multitenant network virtualization. The Cisco Application Policy Infrastructure Controller (APIC) is the heart and brains of the Cisco ACI solution. Cisco APIC offers a single, robust and well documented API to programmatically control all aspects of the system.

Cisco provides a supported and open-sourced Neutron plug-in for Cisco APIC to leverage a Cisco ACI fabric as the back end to implement networking for OpenStack clouds. The Cisco ACI plug-in for OpenStack brings many benefits for both OpenStack and fabric administrators in terms of performance, high availability, visibility and simplified operations.

This document provides a detailed description of the Cisco ACI OpenStack plug-in architecture. The document is for cloud Architects, OpenStack and Cisco ACI fabric administrators. The document assumes previous knowledge of Cisco ACI and at least basic notions of OpenStack.

OpenStack and Neutron Overview

OpenStack defines a flexible and modular software architecture for implementing cloud-computing environments, also referred to as SDN data centers in some literature.

OpenStack Nova, also known as OpenStack Compute, defines how to manage multiple physical compute resources as a pool of virtual capacity orchestrating the hypervisor layer. Nova can launch virtual machines (VMs), called instances in OpenStack, that are scheduled across physical compute systems running a hypervisor. These hypervisors are commonly referred to as Nova nodes or Nova compute nodes. The most popular hypervisor supported to implement Nova nodes is Linux Kernel-based Virtual Machine (KVM).

Other important OpenStack components take care of maintaining images used to boot instances (Glance), providing block storage resources to implement persistent volumes (Cinder), or keeping track of administrator's identity and permissions (Keystone).

In most cases, all these projects are implemented as a set of servers that may or not run concurrently on the same machines and communicate between them using a message queue service (typically RabittMQ or Qpid). The general practice is to dedicate multiple servers to run these services in a highly available mode. These servers are called controllers.

OpenStack instances require network connectivity. Networking is a standalone component in the OpenStack modular architecture. The key project for implementing network and security in OpenStack is Neutron. Neutron replaced a former version of the network service called Quantum, introduced with the Folsom release of OpenStack. Before this, networking for OpenStack instances was handled directly from Nova.

Neutron provides a reference implementation to provide many basic and advanced network services, including IP address management (IPAM), Layer2, Layer3, Network Address Translation (NAT), and security services for OpenStack instances. Neutron can also be used to implement load balancing and VPN services.

Neutron Architecture

Neutron is based on a pluggable architecture. The fundamental component is the neutron-server daemon. This server typically run on the OpenStack controller cluster mentioned above, but it can also be installed on dedicated servers. The neutron-server exposes the OpenStack networking REST API, implements a remote procedure call (RPC) service to communicate with the messaging bus, and provides support for various plug-ins. A Neutron plug-in can be described as a collection of Python modules that implements a standard interface, that accepts and receives some standard API calls, and connects with devices

downstream. The neutron-server requires access to a database (Neutron Database), and many plug-ins may also require access to a database for persistent storage as well. In most implementations the neutron-server and the configured plug-ins leverage the same database services available to other OpenStack core components in the controller nodes.

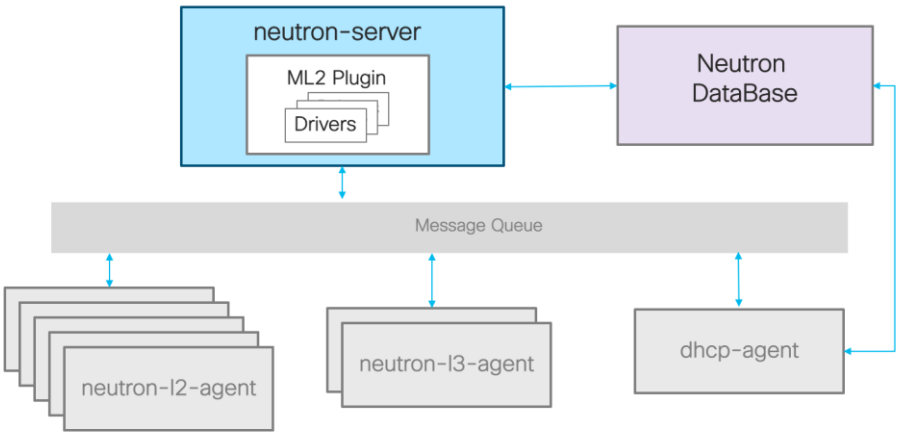
Neutron plug-ins are divided into core plug-ins and service plug-ins. Core plug-ins provide the core Neutron API functionality, which is essentially Layer 2 and IP address management. In many cases, they also provide Layer 3 and security services (such as. security groups, which are explained later in this document). Service plug-ins, on the other hand, are used for things like Load Balancing as a Service (LBaaS), Firewall as a Service (FWaaS) or VPN as a Service (VPNaaS). Multiple plug-ins can be installed concurrently on a single Neutron server.

The core plug-in functionality is largely defined by the Modular Layer 2 (ML2) Neutron plug-in framework. ML2 uses two kinds of drivers that can be configured:

- TypeDrivers, which define how an OpenStack L2 network is implemented. F for instance the driver can be flat, VLAN, VXLAN, GRE, and so on. The TypeDriver keeps track of the encapsulation space to allocate unused segments,. They are configured on the /etc/neutron/plug-ins/ml2/ml2_conf.ini file as type_drivers. Multiple options can be configured at the same time.
- Mechanism Drivers, essentially take the information from a type driver and implement it for a given networking mechanism. For instance, a Linux Bridge mechanism driver can be used to implement a network of type VLAN. They are configured on the /etc/neutron/plug-ins/ml2/ml2_conf.ini as mechanism_drivers. Multiple options can be configured at the same time.

Therefore, apart from installing and running the neutron-server component on the controller nodes, common Neutron implementations also require installing plug-ins. These in turn typically rely on a set of agents that may run on the Nova nodes as well as on the controller nodes. The reference implementation for the instance implements a Layer 2 agent (neutron-l2-agent) that must run on every Nova compute node, and Layer 3 agents (neutron-l3-agent) that run on dedicated network nodes to route between different subnets.

Figure 1: Neutron Architecture



IP address management in Neutron is provided by a DHCP agent, commonly implemented using Dnsmasq, that runs on the Neutron server for every Neutron network. The Neutron server also runs a metadata service to provide instances to additional information, such as inserting SSH keys or running cloud-init scripts.

Neutron Network Model

Neutron provides a very simple networking model with the following key concepts:

- Network: An isolated Layer 2 segment, similar to a VLAN in traditional networking.
- Subnet: A block of addresses associated with a Neutron network. Multiple subnets can be associated with a single network.
- Port A connection point to attach a device, typically the virtual NIC of an instance, to a virtual network.
- Router: Implements connectivity between different Neutron networks and subnets. Routers also can connect to external networks to route outside of the OpenStack cloud.

In the reference implementation, networks are implemented by the L2 agent using one of the configured drivers, such as Linux Bridge, or OVS. The Neutron routers in that case are implemented on the Layer 3 agent running on the network nodes, which are dedicated servers that route between Neutron subnets, and between Neutron subnets and external networks. When instances on the Neutron networks need to communicate with outside networks, NAT may be required. NAT also is implemented on the network nodes, which should also be deployed with redundancy.

The common process when an OpenStack instance is required involves Neutron. The Nova controller calls the Neutron server to ask for a port be created for the new instance on the selected Nova node. The instance then boots on the given nova node. Neutron then creates the port and attaches it to the instance and notifies the DHCP service of the new port. When the instance is booting and connected to the neutron port, it can send a DHCP request for its address, and the DHCP service on the network node replies with the instance's assigned IP address.

Neutron and Security

Neutron security is implemented using *security groups*. Security groups are definitions of Layer 2-to-Layer 4 filters that define the traffic allowed to and from instances. Instances always belong to a default security group and can belong to multiple security groups, thus allowing cloud administrators to define granular and modular security policies.

Security groups by default also implement antispoofing techniques. For instance, by default, traffic from an instance is only allowed if it is sourced by the MAC address and IP address given to the instance by the Neutron DHCP process.

Challenges with OpenStack Neutron

The Neutron reference implementation provides a functional networking solution for OpenStack. However, it presents various challenges:

- Implementing Layer2 and Layer3 services over existing network infrastructure is extremely complicated and limited to basic provisioning only.
- To overcome the previous point, overlay technologies can be used (for example, GRE or VXLAN) for implementing Layer 2 or Layer 3 services. However, with the reference implementation, the overlay technologies lack scale and limit visibility for network administrators. Overlay technologies can also introduce performance challenges with servers not supporting the hardware offload of the tunnel encapsulation.
- Communication between OpenStack instances and external endpoints (EPs) must be routed through the Neutron servers, which may become a performance bottleneck and limit high availability.
- NAT/SNAT functions run centralized on Neutron servers, representing a performance choke point and lacking high availability solutions.
- Network operators have limited to no visibility into OpenStack resources.

Cisco ACI Overview

Cisco (ACI) is the Cisco next-generation SDN platform that provides full network automation through REST APIs. Cisco ACI is based on a new declarative network security policy model that focuses on applications and extends to multiple hypervisors, containers, and bare metal servers. It also offers line-rate routing and switching capabilities across different server domains.

With Cisco ACI, it is easy to provide end-to-end VXLAN tunneling and automation of Layer 2-to-Layer 7 network policies across multiple geographically dispersed and scalable data centers and public clouds.

Cisco ACI comes with an integrated Graphical User Interface (GUI) that simplifies operating and monitoring the platform in a multitenant fashion providing Fault Configuration Accounting Performance Security (FCAPS) capabilities.

Cisco ACI fabric has three main components:

- Cisco APIC controller
- Nexus 9000 Series switches
- Cisco ACI software

Cisco APIC

Cisco APIC is a clustered system providing image management, bootstrap and policy configuration. Cisco APIC is the Cisco ACI brain, which acts as a single, redundant, and distributed point of control. It is important to understand that in the Cisco ACI fabric only Nexus 9000 Series switches are responsible for routing network packets. Cisco APIC is never in the data path.

The Cisco APIC cluster consists of three or more nodes running actively and sharing the policy knowledge base. A concept known as *data sharding* is used with the Cisco APIC. The basic idea behind sharding is that the data repository is split into multiple database units, known as *shards*. Only one Cisco APIC controller is elected the leader for a specific shard, while the other Cisco APIC controllers host a shard replica always consistent with the leader.

The Cisco APIC consists of several control functions, including:

- Policy Manager: A distributed policy repository responsible for the definition and deployment of the policy-based configuration of the Cisco ACI fabric.
- Topology Manager: Maintains up-to-date Cisco ACI topology and inventory information.
- Observer: The monitoring subsystem of the Cisco APIC, serving as a data repository of the Cisco ACI operational state, health and performance.
- Boot Director: Controls the booting and firmware updates of the spine and leaf switches, as well as the Cisco APIC controller elements.
- VMM Manager: Acts as an agent between the policy repository and a hypervisor and is responsible for interacting with hypervisor management systems such as VMware vCenter, OpenStack, Microsoft SCVMM, Red Hat Enterprise Virtualization Manager, and others.
- Event Manager: A repository for all the events and faults initiated from the Cisco APIC or the fabric nodes.

Cisco ACI uses a fundamentally innovative approach to distribute policies across network elements. This approach is called *declarative* control. Declarative control dictates that each object of the system is asked to achieve a desired state, and promises to reach this state, without being told precisely how to do so. This model requires that the objects are smart enough to accept the promise and implement the platform-specific configuration to achieve the requested state.

Declarative control was created to overcome scale and flexibility limitations of the imperative model. In this latter model, the controller instructs the object nodes with the specific step-by-step configuration that should be implemented.

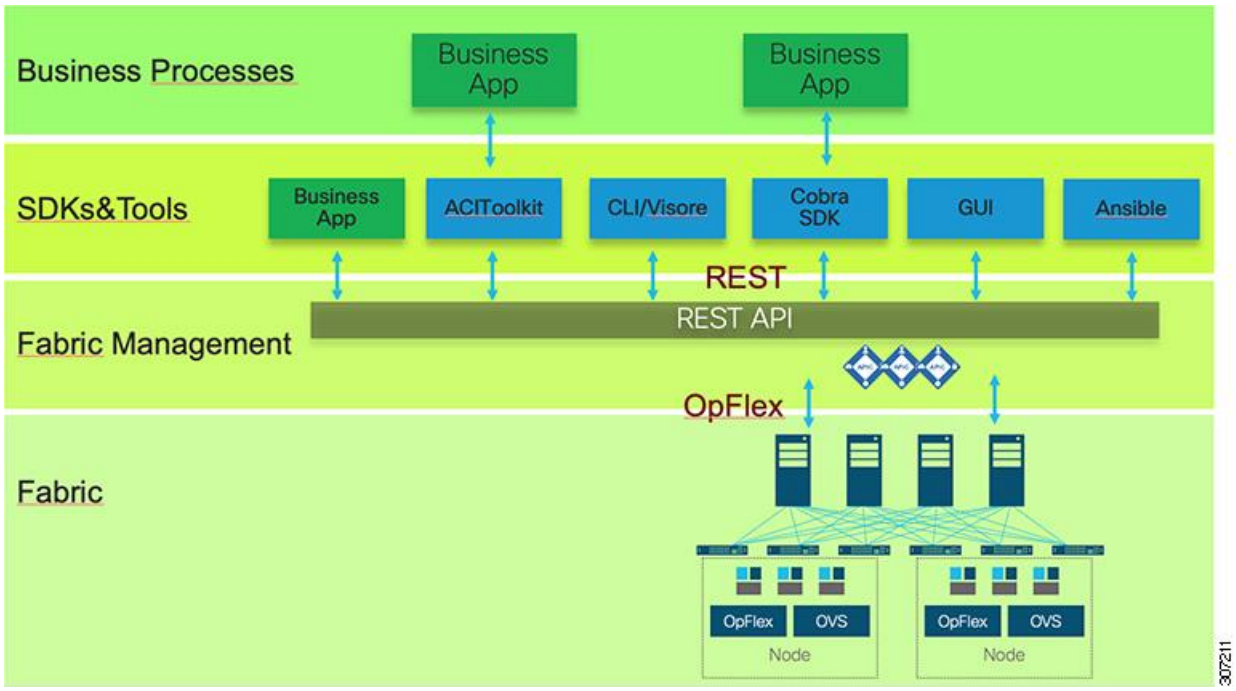
Cisco ACI uses the OpFlex mechanism protocol to transfer abstract policy from Cisco APIC to the network policy enforcement nodes. OpFlex is an open and extensible policy protocol for transferring abstract policy in XML or JavaScript Object Notation (JSON) between a network policy controller.

Cisco APIC is capable of provisioning policies to OpFlex agents that may run in multiple node types. Example of node types supporting OpFlex agents are:

- Nexus 9000 Series switches
- External routers, i.e. Cisco ASR9000, Cisco ASR1000, Cisco Nexus 7000, Cisco CSR1000v
- VMware ESXi hosts
- OpenStack nova compute nodes
- Microsoft Hyper-V hosts
- Container hosts

Cisco APIC provides RESTful APIs. Cisco ACI administrators can leverage the RESTful APIs over HTTP(s) with XML and JSON encoding bindings. As shown in Figure 2, there are a number of tools and SDKs built on top of the REST APIs.

Figure 2: Cisco APIC End-to-End Programmability



Cisco Nexus 9000 Series Switches

The Cisco ACI fabric is a leaf and spine architecture, with each leaf connecting to every spine, but with no cross connects at either layer (technically known as a *bipartite graph*).

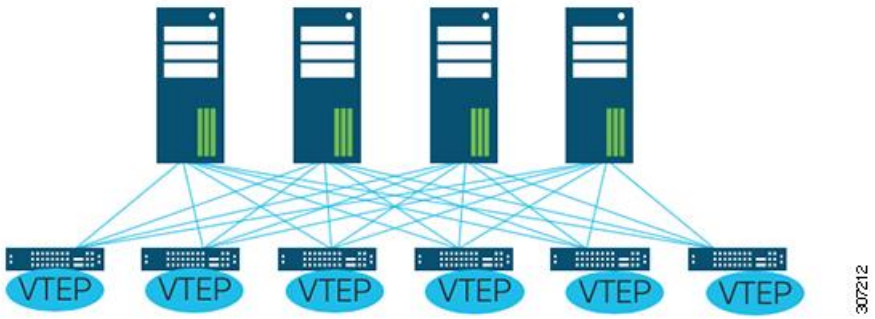
The physical switching layer in the Cisco ACI platform is based on the Cisco Nexus 9000 Series switching platform. The fabric provides consistent, low-latency forwarding across high-bandwidth links. Traffic with a source and destination on the same leaf is handled locally, with all other traffic being forwarded between ingress and egress leaf through the spine layer.

The spine layer of the Cisco ACI infrastructure can be either in a modular or fixed format and offers 40/100/400G ports supporting QSFP28 bidirectional transceivers.

The leaf layer is in fixed format and offers 100Mb-to-400G port capacity.

You can find more information on Nexus 9000 Series switches on Cisco.com.

Figure 3: Cisco ACI Fabric—Nexus 9000Series



At the access layer, each Cisco ACI Leaf port can dynamically expose the overlay network on the specific packet encapsulation expected by the type of endpoint that it is connected to.

This means that heterogeneous endpoint types from different hypervisors, bare-metal servers, or other network elements can all coexist in the same network overlay, while the Nexus 9000 Series leaf switches take care to normalize the external encapsulation into a unique VNID for each network overlay. In other words, multiple VLANs or VXLAN encapsulations at the access level can be mapped dynamically by Cisco ACI into the same network segment unit, which in Cisco ACI is called an endpoint group (EPG).

In the Cisco ACI fabric each leaf switch acts as a hardware virtual tunnel endpoint (VTEP), and all traffic between any VTEP pair is encapsulated into a VXLAN tunnel. Cisco ACI VTEPs could be dispersed in multiple data centers through the Cisco ACI Multipod or Cisco ACI Multi-Site technologies.

Nexus 9000 Series switches have a flexible architecture and can run both NXOS or Cisco ACI software.

When in Cisco ACI mode, the switches learn policies from Cisco APIC through OpFlex. To optimize resources, policies are only configured where required. For example, Cisco ACI uses the concept of pervasive gateway. This means that—when Cisco ACI is used as gateway for the endpoints—the routing device is always one Layer 2 hop away from the endpoint because the routing functions are distributed in a pervasive manner. To optimize resources of the Cisco ACI leaf switches, Cisco APIC only distributes the specific routing policies to the required leafs where endpoints belonging to the particular network segments are discovered as directly connected to the leaf.

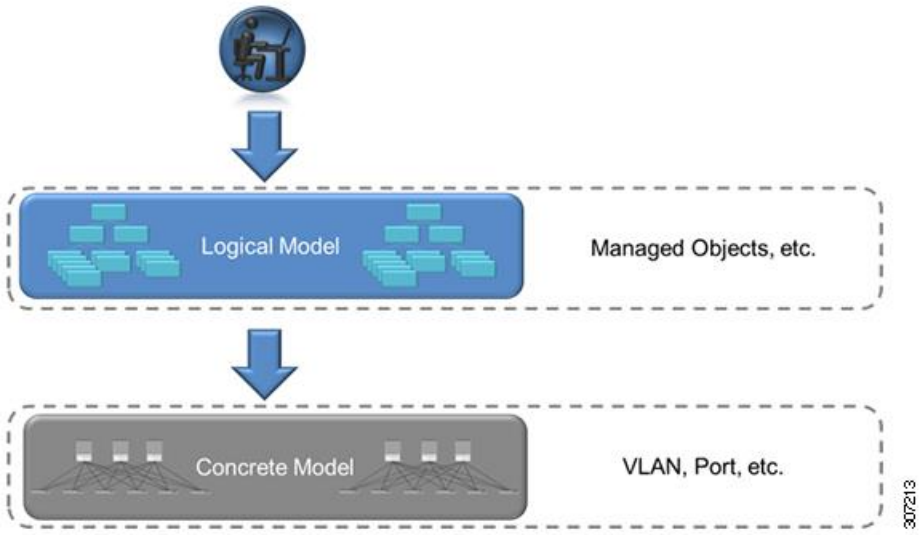
Cisco ACI Software and Policy Model

When deployed in Cisco ACI mode, Cisco APIC controller and Cisco Nexus 9000 Series switches run Cisco ACI software. Cisco ACI software maintains a complete representation of the operational state of the system, with the model being applied in a uniform manner across the fabric, services and system behavior as well as virtual and physical devices attached to the network.

The Cisco ACI policy model provides a convenient way to specify application requirements, which Cisco APIC then renders in the network infrastructure. The policy model consists of several constructs such as tenants, contexts, bridge domains, and endpoint group. When a user or process initiates an administrative change to an object within the fabric, that change is first applied to the Cisco ACI policy model (logical model) and then applied to the actual managed endpoint (concrete model).

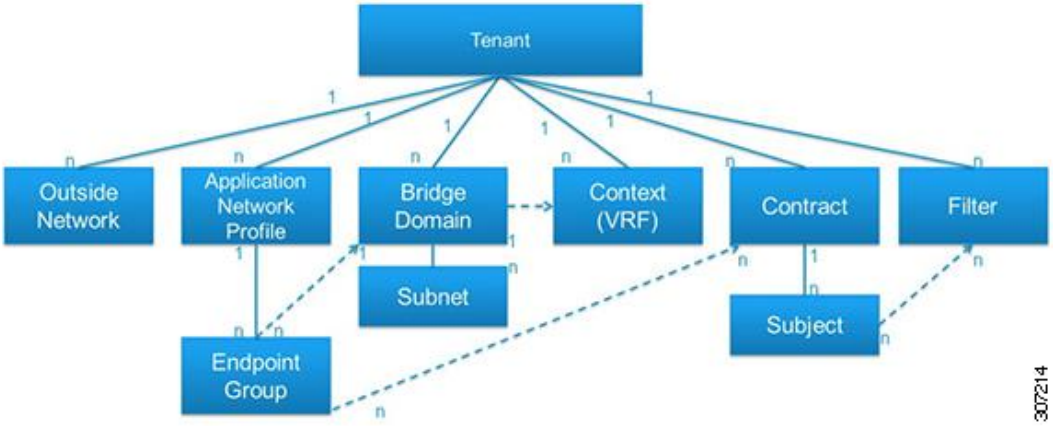
Figure 4 shows how the two models work together.

Figure 4: Logical / Concrete Model



All physical and logical components of the Cisco ACI fabric are represented as a hierarchical *Management Information Tree* (MIT). Some of the key components contained within the MIT are shown in Figure 5.

Figure 5: Key Cisco ACI Policy Constructs



You can find more information about the Cisco ACI Policy Model in *Cisco Application Centric Infrastructure Fundamentals* on Cisco.com .

The relevant managed objects for the context of this document are defined in the rest of this section.

Tenants

A tenant is essentially a container, used to house other constructs and objects in the policy model (such as contexts, bridge domains, contracts, filters and application profiles). Tenants can be completely isolated from each other, or they can share resources. A tenant can be used to define administrative boundaries—administrators can be given access to specific tenants only, resulting in other tenants being completely inaccessible to them.

VRF Objects

A virtual routing and forwarding (VRF) object is used to define a unique Layer 3 forwarding domain within the fabric. One or more VRFs can be created inside a tenant. Because each VRF defines a separate Layer 3 domain, IP addresses residing within a context can overlap addresses in other VRFs.

Bridge Domains and Subnets

A bridge domain is a construct used to define a Layer 2 boundary within the fabric. A bridge domain can be viewed as somewhat similar to a regular VLAN in a traditional switching environment. However, bridge domains are not subject to the same scale limitations as VLANs and have several enhancements, such as improved handling of ARP requests and no flooding behavior by default. A bridge domain always associates with one VRF.

A subnet defines the gateway(s) used within a given bridge domain. This gateway will typically be used by hosts associated with a bridge domain as their first-hop gateway. Gateways defined within a bridge domain are pervasive across all leaf switches where that bridge domain is active.

EPGs

The EPG is one of the most important objects in the policy model and is used to define a collection of endpoints. An endpoint is a device connected to the fabric (either directly or indirectly) and has an address, a location, and other attributes. An EPG always associates with one BD.

Endpoints are grouped together into an EPG, where policy can be more easily applied consistently across the Cisco ACI fabric. Endpoints belonging to one EPG can by default talk to each other without constraints.

Application Profiles

The application profile is the policy construct that ties multiple EPGs together. An application profile contains as many EPGs as necessary to logically relate to the capabilities provided by an application.

Contracts

A contract is a policy construct used to define the communication between different EPGs. Without a contract, no communication is possible between those EPGs. Within an EPG, a contract is not required to allow communication because this is always allowed. Figure 6 shows the relationship between EPGs and contracts.

Figure 6: EPGs and Contracts

An EPG provides or consumes a contract (or provides and consumes different contracts). For example, EPG “Web” in Figure 6 will provide a contract that EPG “App” will consume. Similarly, EPG “App” provides separate contracts that are consumable by the “Web” and “DB” EPGs.

Subjects

A subject is a construct contained within a contract and that typically references one or more filters.

Filters

A filter is a Layer 4 network rule specifying fields such as protocol type and port. It is referenced within a contract to define the communication allowed between EPGs in the fabric.

A filter contains one or more filter entries that specify the rule.

Host Protection Profiles

Host protection profiles are the representation in Cisco ACI of network security policies implemented in the compute nodes where OpFlex agent is running. Typically, those policies are enforced in a distributed fashion directly on a virtual switch on the same host.

More Information About Cisco ACI

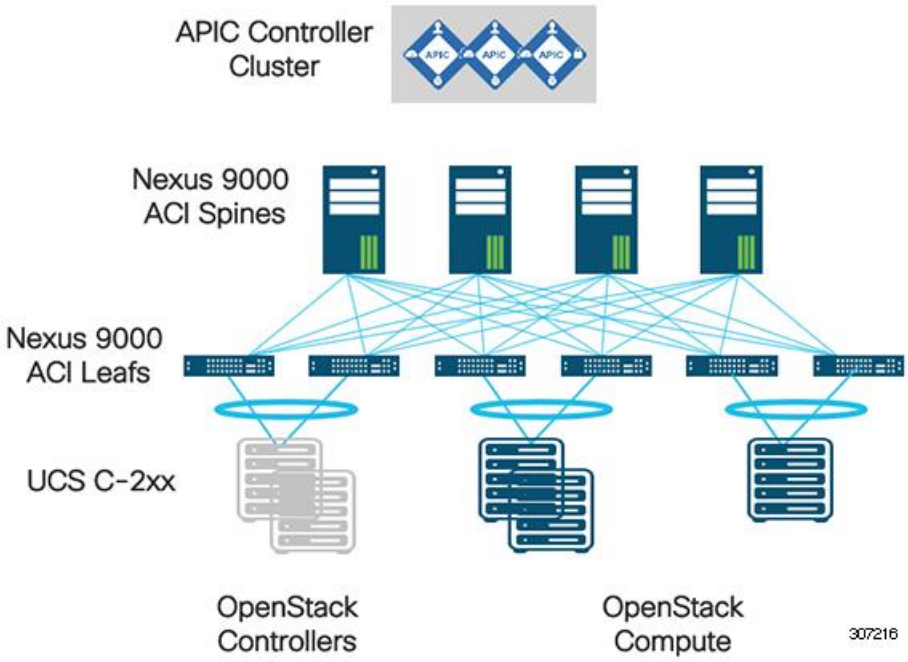
You can find detailed information about Cisco ACI on the Cisco Application Centric Infrastructure page on Cisco.com.

OpenStack and Cisco ACI

A modern software-defined data center is best built with a modern network design architecture based on a leaf-spine topology with maximum bisectional bandwidth. An OpenStack cloud environment typically requires high bandwidth not only for user workloads, but also for storage, live migration operations, image management, and other tasks.

Cisco ACI delivers a best-of-breed leaf-spine topology with a single point of management. The Nexus 9000 Series family of switches offers nonblocking switching architectures with ideal connectivity options for OpenStack nodes. These include 10GE, 40GE, or 100GE interfaces for Nova or storage nodes, 40GE, 100GE, or 400GE uplinks from the top of the rack switches and affordable 1GE interfaces for server management functions, such as connecting Cisco UCS Cisco Integrated Management Controller (CIMC) interfaces.

Figure 7: Controller Node Connection



The OpenStack infrastructure requires connectivity for various services that run on dedicated EPGs on the Cisco ACI fabric, including Preboot Execution Environment (PXE), management, storage, and Cinder. Cisco APIC programmability is perfect to set up the fabric for an OpenStack environment, providing connectivity and security for all required services mapping EPGs through physical domains. As seen in Figure 7, controller nodes should be connected to different leafs than Nova compute nodes to implement a more resilient and easy-to-operate infrastructure. We address physical design considerations later in this document.

Cisco ACI Unified Plug-in for OpenStack

In addition to infrastructure connectivity services, OpenStack clouds also need to provide networking for tenant instances to communicate between themselves and external networks. Customers can use any given OpenStack networking solution running on top of Cisco ACI. However, Cisco offers a Cisco ACI OpenStack plug-in to perform these functions.

The Cisco ACI OpenStack Neutron Plug-in provides effective and affordable solutions to the ordinary Neutron challenges outlined earlier in this document. The Cisco ACI OpenStack Neutron Plug-in is only supported with commercially supported OpenStack distributions. See the release notes for specific details of the distributions and versions supported.

The plug-in can be deployed in two per-node configuration options that are not mutually exclusive:

- OpFlex Mode (OpFlex-ovs): In this option, Cisco APIC controls the upstream Open vSwitch (OVS) running on each Nova compute node by using the OpFlex protocol. This requires installing Cisco OpFlex and OVS agents running on each of the compute nodes. This deployment option implements a virtual machine manager (VMM) on Cisco APIC to provide the fabric administrator maximum visibility of the OpenStack cloud. When choosing the OpFlex mode, the Cisco ACI OpenStack Plug-in replaces the Neutron node datapath enabling fully distributed Layer 2, anycast gateway, DHCP, metadata optimization, distributed NAT, and floating IP enforcement.
- Non-OpFlex Mode: In this option, Cisco APIC only programs the physical fabric and treats OpenStack tenant traffic as part of P\physical domains (PhysDoms). This option can leverage SR-IOV or OVS-DPDK on the compute nodes and does not require installing Cisco agents on the nodes.

Regardless of the deployment model (OpFlex-ovs or non-OpFlex), integrating Cisco ACI with OpenStack by using the Cisco ACI OpenStack plug-in allows creation of networking constructs in Cisco ACI to be driven from OpenStack using standard Neutron calls. At the same time, the integration provides visibility within Cisco (APIC) down to the level of the individual VM instance. The Cisco ACI Integration Module (AIM) automatically enforces this policy synchronization.

Regardless of the deployment option you choose, the Cisco ACI OpenStack Neutron plug-in implements two different networking models for OpenStack: Neutron ML2 and Group Based Policy. Before Release 2.3, the Cisco ACI OpenStack Plug-in was shipped as two independent plug-ins, one for each of the networking models. Starting with Release 2.3, Cisco has shipped a single plug-in, referred to as the unified plug-in. For the rest of this document we will cover this unified plug-in exclusively, and refer to it as the Cisco ACI OpenStack Plug-in.

ML2 and GBP Networking Models

The Cisco ACI OpenStack plug-in leverages Cisco APIC to implement OpenStack tenant networking and security services. To do so, it maps Neutron concepts such as networks, subnets, and routers to Cisco ACI concepts such as EPGs, bridge domains and contracts. The Group Based Policy (GBP) networking model is very similar to the Cisco ACI policy model. It is important to understand that with the Cisco ACI unified plug-in for OpenStack, you can use both ML2 and GBP models on a single plug-in instance' however only one of them must be used for any given OpenStack project. This means that potentially an OpenStack administrator could define a subset of OpenStack projects using the ML2 Neutron model, and another set of projects using the GBP model. A single project should not mix ML2 and GBP configurations.

ML2 Networking Mapping to Cisco ACI

In the Cisco ACI Plug-in for OpenStack naming convention, we call the ML2 deployment model the way of leveraging standard Neutron calls to provision network policy constructs such as network, router, and security groups.

When using ML2 deployment model, an OpenStack administrator already using Neutron through Horizon, Heat, CLI, or REST API calls transparently transitions to operate an OpenStack integrated with Cisco ACI.

With the ML2 standard Neutron model, the following mapping happens as shown in Table 1.

Table 1: ML2 Deployment Model—OpenStack to Cisco ACI Mapping

OpenStack/Neutron Object	Cisco APIC Object
Project	Tenant and application profile name. Note: the name of the Tenant in APIC will be the UUID of the OpenStack project. The name of the project will be configured under the alias field of the Cisco ACI tenant. A single APN per OpenStack project is created with the name OpenStack.
Network	EPG and bridge domain. Note: The name of the EPG and the bridge domain in Cisco APIC is the UUID of the OpenStack Neutron network. The name of the Neutron network is configured under the alias field of the Cisco ACI objects.
Subnet	Subnet.
Router	Contract, consumed and provided by any EPGs corresponding to the Neutron networks connected to the router. Note: the name of the contract in APIC is the UUID of the OpenStack router. The name of the OpenStack router is configured under the alias field of the Cisco ACI contract. The contract is defined with a filter to allow any IP traffic. Policy enforcement is configured with security groups.
Security Group	Host protection profile (HPP) policies. Note: the name of the HPP in Cisco APIC is the UUID of the OpenStack security group. HPPs are implemented as stateful firewall rules programmed on Open vSwitch (OVS). This is only available when running in OpFlex-ovs mode.

While not listed in Table 1, a VRF is also created on the Cisco ACI tenant corresponding to an OpenStack project. The bridge domains and EPGs that correspond to Neutron networks of that OpenStack project are associated to the tenant VRF.

With ML2 model, standard OpenStack security groups are used. Those security groups are rendered in Cisco ACI as HPP objects. When OpFlex agent is used to control OVS, those policies are pushed from the Cisco APIC to OVS rules through OpFlex protocol. Otherwise, standard OpenStack Neutron API calls will be leveraged from the OpenStack controller to deploy the rules down to the nodes using IP tables (for instance, when using OVS-DPDK in non-OpFlex model).

GBP Networking Mapping to Cisco ACI

The GBP model for OpenStack provides an intent-driven declarative policy model that aims to abstract the application segmentation from the network layer. GBP abstractions for OpenStack are closely aligned with the declarative policy model used in Cisco ACI.

The following mapping happens between GBP and Cisco ACI objects as shown in Table 2.

Table 2: GBP Deployment Model—OpenStack to Cisco ACI Mapping

OpenStack/GBP Object	Cisco APIC Object
Project	Tenant and application profile name. Note: The name of the tenant in Cisco APIC is the UUID of the OpenStack project. The name of the project is configured under the alias field of the Cisco ACI tenant. A single APN per OpenStack project is created with the name OpenStack.
L3 Policy	VRF. Note: The name of the VRF in Cisco APIC is the UUID of the OpenStack GBP L3 Policy. The name of the GBP L3 Policy is configured under the alias field of the Cisco ACI VRF.
L2 Policy	Bridge domain and associated subnet Note: The name of the bridge domain and subnet in Cisco APIC is the UUID of the OpenStack GBP L2Policy and CIDR. The name of the OpenStack objects are configured under the alias field of the Cisco ACI corresponding objects.
Policy Group	EPG. Note: The name of the EPG in Cisco APIC is the UUID of the OpenStack Policy Group. The name of the OpenStack Policy Group is configured under the alias field of the Cisco ACI EPG.
Policy Ruleset	Contract. Note: The name of the contract in APIC is the UUID of the OpenStack Policy Ruleset. The contracts are implemented as stateless rules programmed on the required Cisco ACI leafs as well as on Open vSwitch (OVS) if OpFlex-ovs is being used.

GBP offers a policy API through multiple OpenStack interfaces, including Horizon extensions (openstack-dashboard-gbp), Heat (openstack-heat-gbp), and CLI (openstack-neutron-gbp). These are installed with the Cisco ACI Plug-in for OpenStack. GBP was designed to act as a layer on top of Neutron.

While it is possible to use OpenStack security groups as in the ML2 model, with GBP you can also control Cisco ACI contracts and provide or consume those by policy groups.

GBP Policy Rulesets are enforced on the OpenStack nodes as OpenFlow rules programmed on OVS (when using OpFlex-ovs), but also on the Cisco ACI leaf switches as if they were standard Cisco ACI contracts.

With GBP note the following:

- Multiple groups of endpoints can be assigned to the same subnet (GBP L2 policy).
- Multiple VRFs can be assigned to the same OpenStack project.

GBP introduces a policy model to describe the relationships between different logical groups or tiers of an application. The primitives have been chosen in a way that separates their semantics from the underlying infrastructure capabilities. Resources can be public or local to a specific tenant.

Figure 8 and Table 3: describe the GBP relationship model.

Figure 8: The GBP Relationship Model

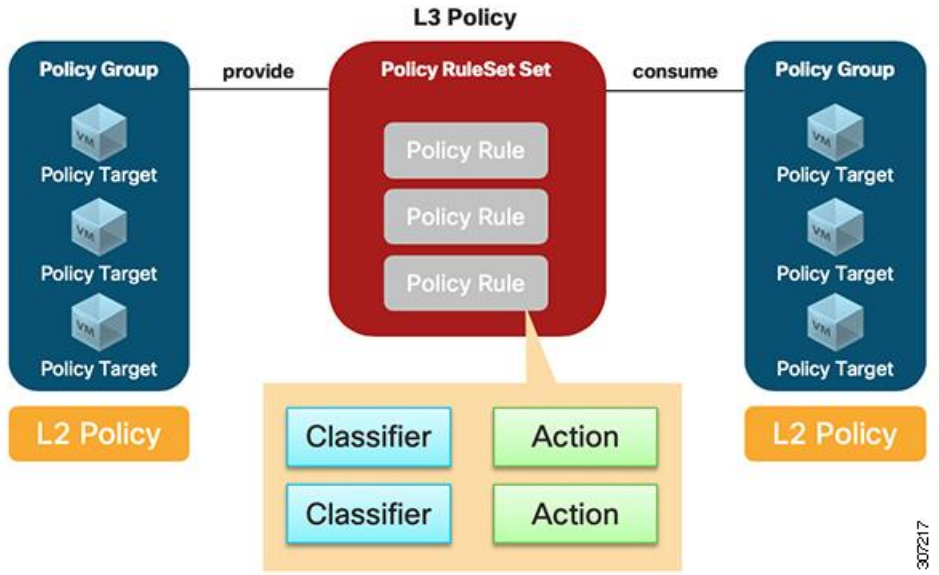


Table 3: The GBP Relationship Model

OpenStack/GBP Object	Description
Policy Target	Policy target is an individual network endpoint (generally a NIC). In other words, this is a basic addressable unit in the architecture.
Policy Group	Policy targets with the same properties are organized into policy groups. A policy group is the fundamental primitive of GBP. Policy groups offer an infrastructure-agnostic grouping construct without specifying any network semantics in the same way as a broadcast. Each group models its dependencies by declaring the rule sets that it provides to groups as well as rule sets that it consumes.
Policy Classifier	A means of filtering network traffic, including protocol, port range, and direction (in, out, or bidirectional).

Policy Action	An action to take when a rule is applied. The supported types include allow and redirect.
Policy Rules	Classifiers paired with actions.
Policy Rule Sets	Policy Rule Sets contain several policy rules. Rule Sets can be nested through parent-child relationships.

The apic_aim ML2 Plug-in

OpenStack ML2 is a framework provided by OpenStack Networking to allow utilization of one or multiple Layer 2 networking technologies. Drivers within ML2 implement extensible sets of network types and of mechanisms for accessing networks of those types.

Cisco ACI provides the apic_aim ML2 plug-in, which offers the OpFlex type driver and the apic_aim mechanism driver. The OpFlex type driver offers seamless VLAN or VXLAN encapsulation to the compute node. Encapsulation type can be defined at installation time. The OpFlex type driver also supports hierarchical port binding (HPB). If deployed in non-OpFlex mode, then it is also possible to use the VLAN type driver. The apic_aim mechanism supports both Neutron and GBP modes for deployment of Cisco ACI policy constructs.

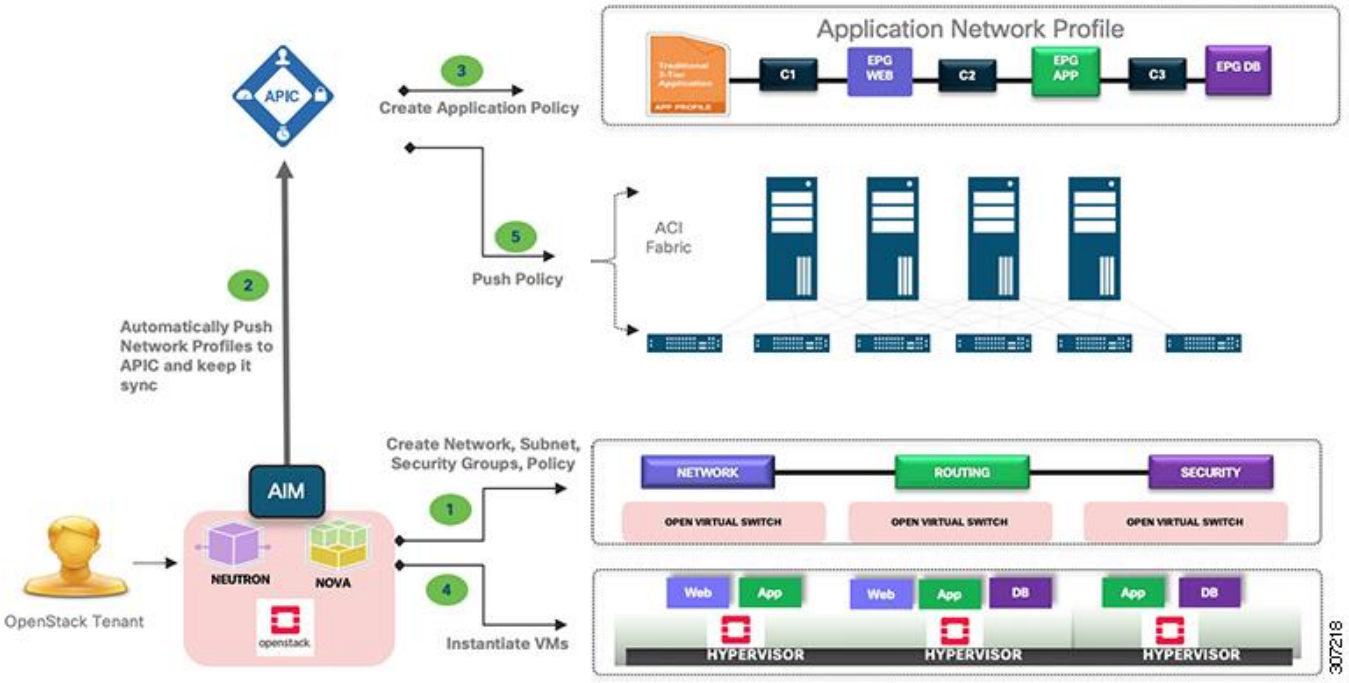
Cisco ACI Integration Module (AIM)

The Cisco AIM is installed on the OpenStack controller nodes and is responsible to configure Cisco ACI through a REST API call based on the defined OpenStack policy model. The ML2 mechanism driver interacts with the Cisco AIM daemon, which talks to the Cisco APIC controller to configure policies. Cisco AIM also stores the Cisco ACI policy model configuration into the Cisco ACI Integration database (AIDB).

Cisco AIM continuously synchronizes with Cisco APIC using the Cisco APIC Integration Daemon (AID).

Figure 9 shows the operational workflow of provisioning network policies with Neutron or GBP calls on the OpenStack controller to the Cisco APIC controller through REST API calls made by the Cisco AIM daemon.

Figure 9: Cisco ACI Integration Module Workflow



1. The OpenStack tenant administrator configures the OpenStack networks through standard Neutron calls or GBP calls using CLI, Heat, Horizon, or REST API calls.
2. The ML2 plus mechanism driver translates the networks created into Cisco AIM policies. Cisco AIM stores the new configuration into its database and pushes network profiles into Cisco APIC through Cisco APIC REST API calls.
3. Cisco APIC creates related network profiles, for example, Cisco ACI tenants, bridge domains, EPGs, and contracts. If OpFlex agent is installed on the compute nodes, OVS rules are also configured accordingly.
4. OpenStack administrator creates instances and attaches those the previously created OpenStack networks.
5. Cisco APIC is notified that new instances have been spawned. Consequently, Cisco APIC pushes the related policies to the leaf switches where the compute nodes running these VMs are attached.

The following components are part of Cisco AIM:

- aim-aid: This is the main Cisco AIM service. This service is responsible for synchronizing the state between the neutron database and Cisco ACI.
- aim-event-service-polling: This is the event service poling daemon.
- aim-event-service-rpc: This is the RPC service.

The following configuration files are part of AIMCTL.

- aim.conf: This configuration file is used for operational parameters, for example, Cisco APIC access (IP, username, password or cert). This file is also used to set rabbitmq and database access.

- aimctl.conf: This configuration file is used to define different control parameters, for example, enabling distributed metadata and DHCP, defining different VMM domains and physical domains to use.

The **aimctl** CLI Tool

Cisco AIM is usually configured using configuration files and the **aimctl** command. The configuration files are used to set the initial state of Cisco AIM and configure the database, Cisco ACI parameters, and other control parameters.

The **aimctl** CLI tool must be run from the OpenStack controller nodes.

The OpenStack controller nodes are also provided with a Cisco AIM CLI tool.

Cisco AIM database initialization is done using the following command: **aimctl db-migration upgrade head**

Any changes in configuration files need to be loaded in the Cisco AIM database. The following command allows update of the Cisco AIM database based on the newly configured files: **aimctl config update**

To reinitialize the Cisco AIM database the following CLI command is available: **aimctl config replace**

Note that the commands for initializing the Cisco AIM are automatically handled by a OpenStack management tool (for example OpenStack Director or Juju Charms). The **aimctl** tool is typically used for troubleshooting and verification. To have an exhaustive list of **aimctl** commands, you can use the tool **aimctl *option* --help**. you must To update the Cisco AIM configuration file, you must manually enter the **aimctl config update** or **aimctl config replace** command.

Because Cisco AIM takes ownership of the Cisco ACI objects created by OpenStack, those objects must be configured only from the OpenStack administrator.

You can use the **aimctl** tool also to modify parameters of the Cisco ACI created objects when there is no Neutron-specific command to modify them. A typical example is to modify the bridge domain settings, such as flooding settings.

The following example shows a command to disable IP learning on a subnet not belonging to the bridge domain.

First, through **aimctl** command, you can query Cisco ACI for an existing bridge domain:

```
[heat-admin@overcloud-controller-0 ~]$ aimctl manager bridge-domain-find

+--------------------------------------+------------------------------------------+

| tenant_name                          | name                                     |

|--------------------------------------+------------------------------------------|

| prj_b75263c9e55c48cab43b2edaf59e5d9a | net_63bac97c-eb1f-4b9b-a8da-9a0c478789bb |

| common                               | openshift-39_bd_kubernetes-service       |

| prj_b75263c9e55c48cab43b2edaf59e5d9a | net_8f2d85b6-89c5-4e93-8339-267e52a01ac9 |

| common                               | BD-T1-SITE2                              |

| common                               | BD-T1-SITE1                              |

| common                               | default                                  |

+--------------------------------------+------------------------------------------+
```

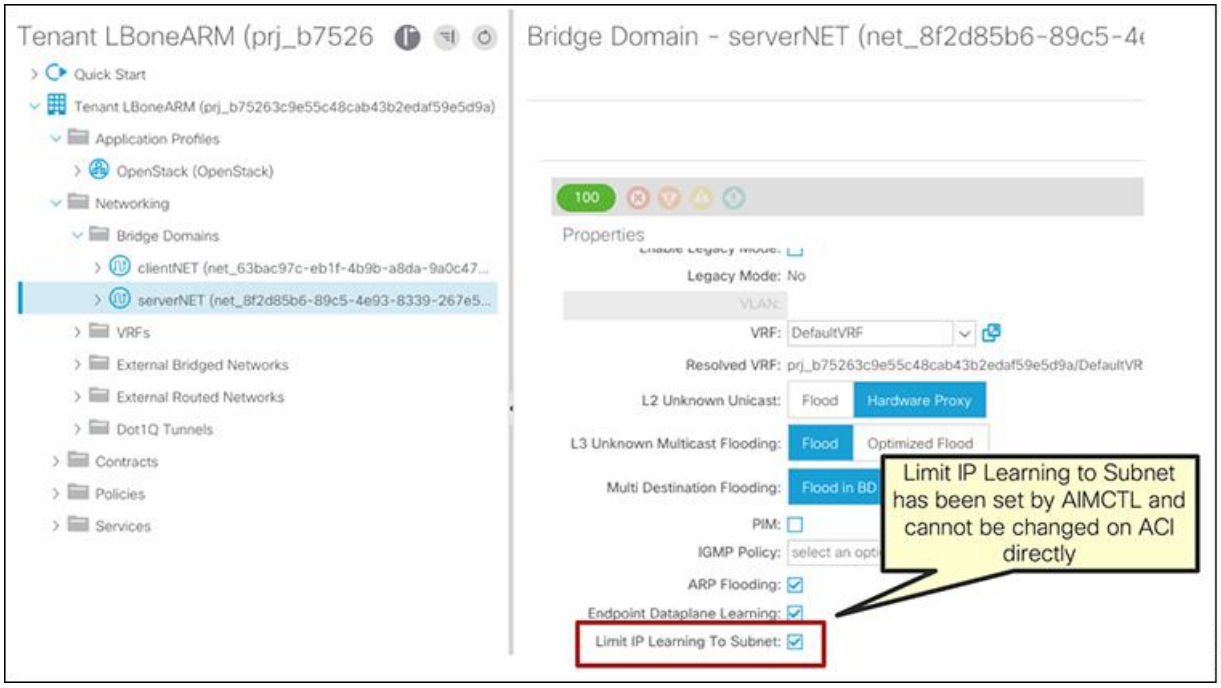
You can then query for settings of a specific bridge domain:

```
[heat-admin@overcloud-controller-0 ~]$ aimctl manager bridge-domain-show
prj_b75263c9e55c48cab43b2edaf59e5d9a net_8f2d85b6-89c5-4e93-8339-267e52a01ac9

+---------------------------+----------------------------------------------------+

| Property                  | Value                                              |

|---------------------------+----------------------------------------------------|

| tenant_name               | prj_b75263c9e55c48cab43b2edaf59e5d9a               |

| name                      | net_8f2d85b6-89c5-4e93-8339-267e52a01ac9           |

| enable_arp_flood          | True                                               |

| display_name              | serverNET                                          |

| limit_ip_learn_to_subnets | True                                               |

| enable_routing            | True                                               |

| ip_learning               | True                                               |

| l2_unknown_unicast_mode   | proxy                                              |

| vrf_name                  | DefaultVRF                                         |

| ep_move_detect_mode       | garp                                               |

| monitored                 | False                                              |

| l3out_names               | []                                                 |

| epoch                     | 2                                                  |

| dn                        | uni/tn-prj_b75263c9e55c48cab43b2edaf59e5d9a/BD-ne|

|                           | t_8f2d85b6-89c5-4e93-8339-267e52a01ac9             |

+---------------------------+----------------------------------------------------+

+---------------+------------------------------------------+

| Property      | Value                                    |

|---------------+------------------------------------------|

| resource_type | BridgeDomain                             |

| resource_id   | 50818618-691c-49a2-940f-497d8e163474     |

| resource_root | tn-prj_b75263c9e55c48cab43b2edaf59e5d9a  |

| sync_status   | synced                                   |
```

```
| sync_message   |                                                |
| faults         | []                                             |
| health_score   | 100                                            |
| id             | 6d417101-53db-4289-a192-281eb8519991           |
| epoch          |                                              7 |
+----------------+------------------------------------------------
```

You can check that the limit_ip_learn_to_subnets on the bridge domain is set to True. This can be also conirmed on the Cisco APIC GUI as shown in Figure 10.

Figure 10: Limit IP Learning on the Bridge Domain

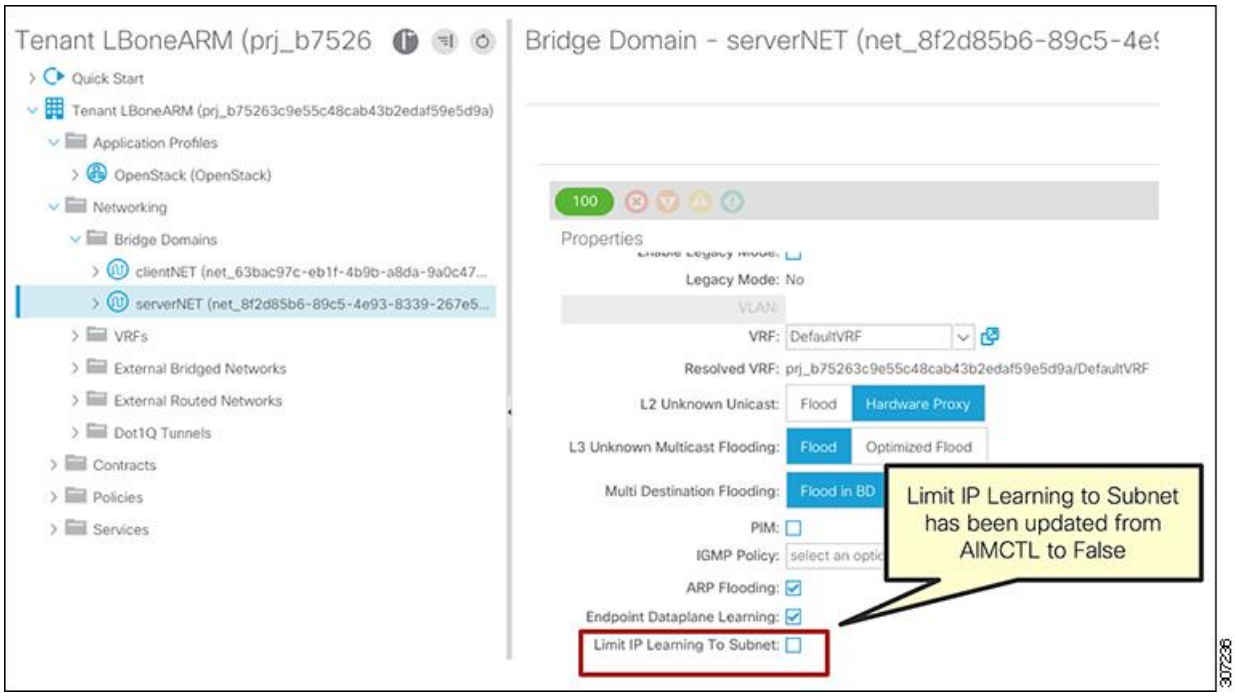


With **aimctl** you can update the Limit IP Learning to Subnet setting to False:

```
[heat-admin@overcloud-controller-0 ~]$ aimctl manager bridge-domain-update
prj_b75263c9e55c48cab43b2edaf59e5d9a net_8f2d85b6-89c5-4e93-8339-267e52a01ac9 --
limit_ip_learn_to_subnets False

+--------------------------+--------------------------------------------------+

| Property                 | Value                                            |

|--------------------------+--------------------------------------------------|

| tenant_name              | prj_b75263c9e55c48cab43b2edaf59e5d9a             |

| name                     | net_8f2d85b6-89c5-4e93-8339-267e52a01ac9         |

| enable_arp_flood         | True                                             |

| display_name             | serverNET                                        |

| limit_ip_learn_to_subnets | False                                           |

| enable_routing           | True                                             |

| ip_learning              | True                                             |

| l2_unknown_unicast_mode  | proxy                                            |

| vrf_name                 | DefaultVRF                                       |

| ep_move_detect_mode      | garp                                             |

| monitored                | False                                            |

| l3out_names              | []                                               |

| epoch                    | 3                                                |

| dn                       | uni/tn-prj_b75263c9e55c48cab43b2edaf59e5d9a/BD-ne|

|                          | t_8f2d85b6-89c5-4e93-8339-267e52a01ac9           |

+--------------------------+--------------------------------------------------+
```

You can also verify this in the Cisco APIC GUI as shown in Figure 11:

Figure 11: Verifying the Limit IP Learning to Subnet Setting in the Cisco APIC GUI

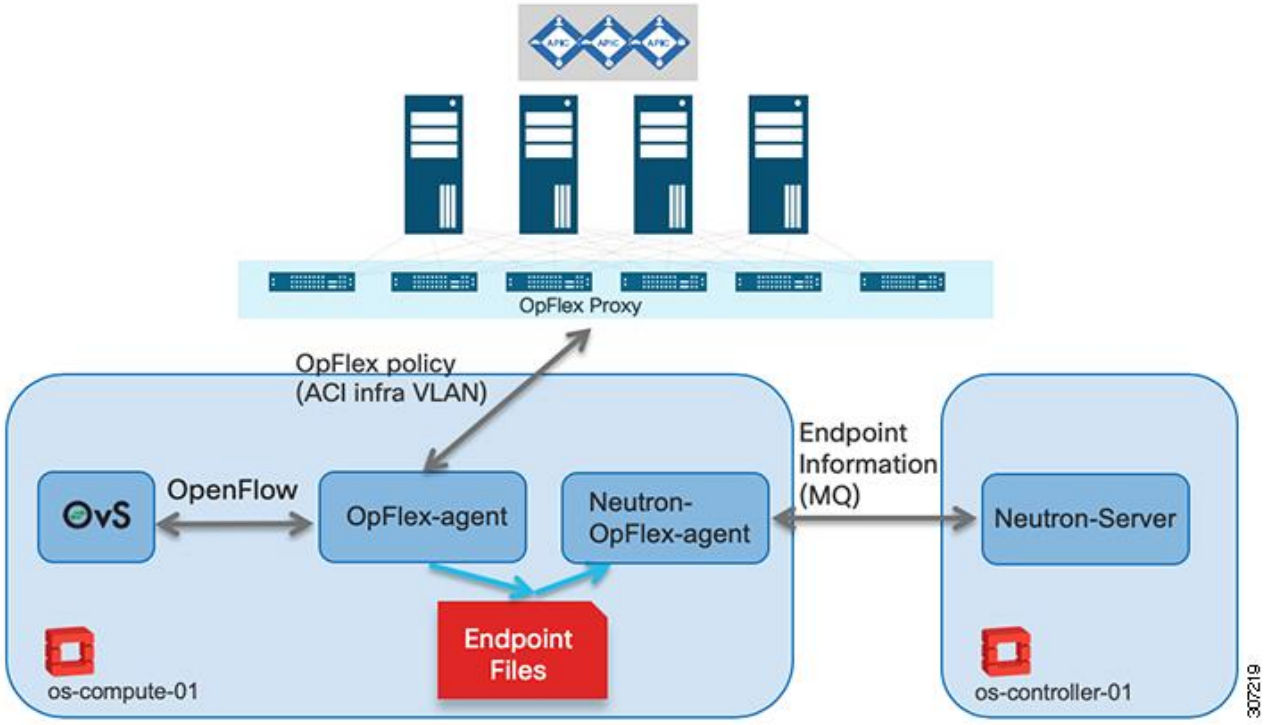


OpFlex Proxy and OpFlex and OVS Agents

Cisco ACI OpenStack plug-in optionally deploys OpFlex and OVS agents, which allow admins to get the visibility into virtual environments from APIC controller.

The Neutron OpFlex Agent runs on both the compute and the controller. It is responsible for communicating with the OpenStack Neutron server. The OpFlex Agent runs on the compute and controller nodes. It is responsible for communicating with the OVS and the OpFlex proxy component of the leaf node where the node is connected to register to the Cisco ACI fabric.

Figure 12: OpFlex and OVS agents



1. Neutron-OpFlex-agent receives updates from Neutron about new endpoints and updates endpoint and service files.
2. OpFlex-Agent runs OpFlex protocol with the OpFlex proxy running on the Cisco ACI leaf. Based on the policies, OpFlex-Agent programs Open vSwitch through OpenFlow.

Other than increasing visibility on Cisco ACI of the policies provisioned for the OpenStack compute node, agents also allow you to configure both VLAN or VXLAN as encapsulation methods between the compute node and the Cisco ACI leaf switch.

If agents are not installed, VLAN encapsulation must be used from the node up to the Cisco ACI leaf. Cisco ACI will then normalize that encapsulation into a VNID of the matched EPG.

OpFlex and PhysDom Deployments

You can deploy the Cisco ACI Plug-in integration with OpenStack in two different ways:

- OpFlex Mode: When the OpFlex and OpFlex OVS agents are deployed for the Cisco ACI integration in the OpenStack compute node.
- PhysDom or non-OpFlex mode: The compute node runs the default OVS agent provided by the OpenStack distribution installed. In this mode, there is no installation of the OpFlex and OpFlex OVS agents on the compute nodes.

Running an OpenStack deployment with both OpFlex and PhysDom nodes is supported. A typical use case to deploy PhysDom nodes is when using data plane acceleration on the nodes, for example, SR-IoV or OVS DPDK. As of January 2019, OpFlex mode does not support the latter, but PhysDom mode does. That means you can have a mix of OpFlex nodes with standard OVS switches, and other PhysDom nodes running SR-IoV and DPDK. Both can belong to the same OpenStack installation.

Using hierarchical port binding, you can even deploy on the same node both OVS networks driven by OpFlex integration and SR-IoV or DPDK networks with PhysDom integration.

Table 4 shows which accelerations can run on the same physical hosts.

Table 4: Combination of Acceleration on the Same Host

Acceleration Mode Possible on the Same Host	Standard OVS with OpFlex	SR-IoV	OVS-DPDK no OpFlex
Standard OVS with OpFlex	Yes	Yes	No
SR-IoV	Yes	Yes	Yes
OVS-DPDK no OpFlex	No	Yes	Yes

Table 5 shows the type of acceleration supported by OpFlex and non OpFlex modes.

Table 5: Acceleration Support for OpFlex and non-OpFlex Modes

Type Driver--Network Type/ Mechanism Driver	VLAN without OpFlex	VLAN with OpFlex	VXLAN with OpFlex
Open vSwitch	Yes	No	No
SR-IoV	Yes	No	No
OpFlex	No	Yes	Yes

Table 6 shows optimization in terms of distributed function of OpFlex compared with non- OpFlex modes in the compute nodes.

Table 6: Optimization Based on the Compute Deployment Mode

Mode / Optimization	Distributed Routing	Distributed NAT	DHCP Optimization	Metadata Optimization	OVS rules or IPTables
OpFlex	Yes	Yes	Yes	Yes	OVS Rules
Non OpFlex	No	No	No	No	IPTables

OpFlex Node Deployment

OpFlex is an open and extensible policy protocol designed to transfer declarative networking policies such as those used in Cisco ACI to other devices. With OpFlex, the policy model native to Cisco ACI can be extended all the way down into the virtual switches running on the OpenStack hosts. This OpFlex extension to the compute node allows Cisco ACI to use OVS to support common OpenStack features such as routing, Source NAT (SNAT) and floating IP in a distributed manner.

All Cisco ACI leaf switches provide an OpFlex proxy service. The OpFlex agents running on the host are connected to the proxy through the Cisco ACI infrastructure network, commonly referred as infra VLAN. The compute nodes are provisioned with a Linux subinterface to communicate with the infra VLAN and to obtain a Cisco APIC-provisioned IP address through DHCP from the Cisco ACI tunnel endpoint (TEP) pool. Once the IP connectivity is established and the OpFlex aware agent can connect to the proxy and query Cisco ACI policies, the leaf effectively becomes an extended Cisco ACI leaf.

The OVS agents communicate to the OpFlex proxy through the infra VLAN of the Cisco ACI fabric.

The OpFlex communication happens through the TCP protocol on port 8009. Data between the OpFlex proxy and agent can be encrypted with SSL, which is enabled by default.

The OpenStack nodes are automatically assigned to the Cisco ACI infra tenant and appear under the default EPG of the access application profile as shown in Figure 13.

Figure 13: OpenStack Nodes in the Cisco ACI Infra Tenant

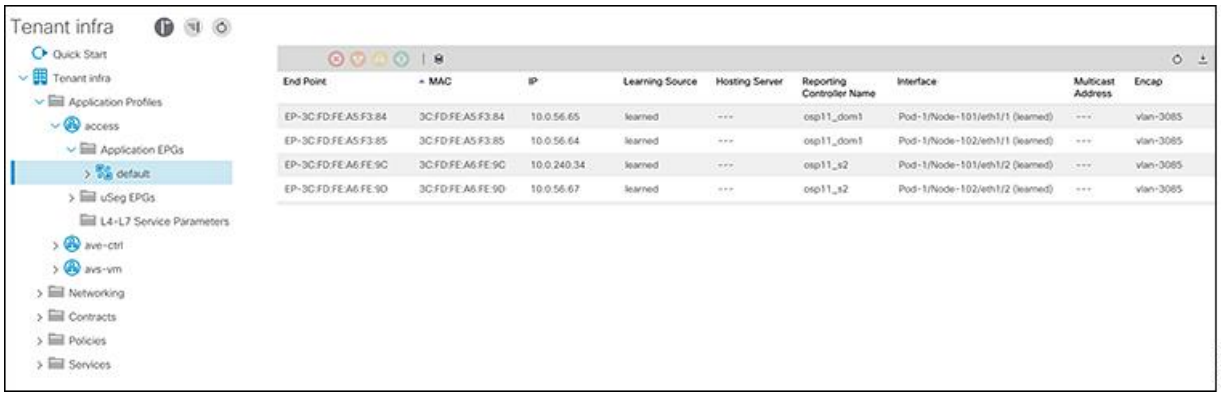
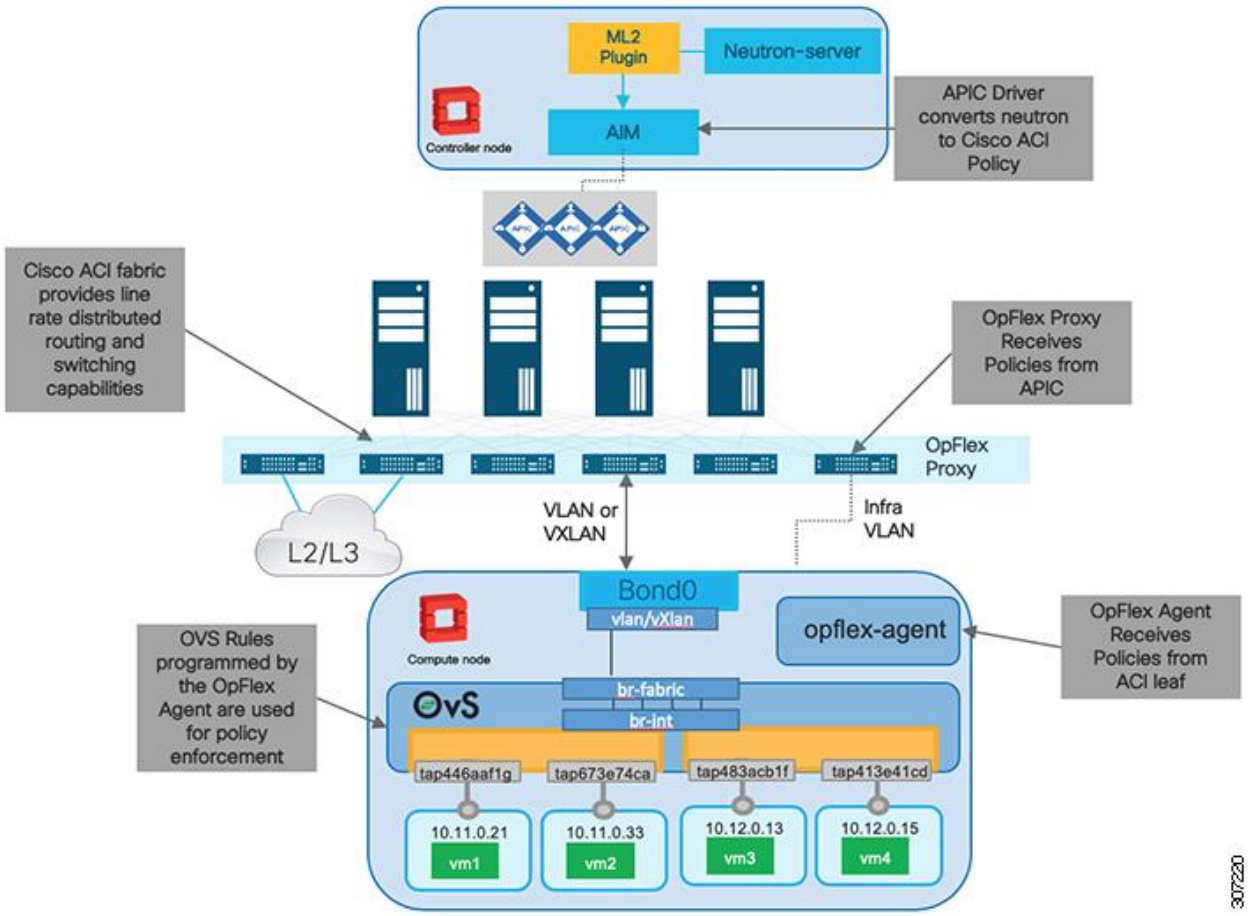


Figure 14 illustrates the typical operation of the OpFlex Cisco ACI OpenStack plug-in.

Figure 14: OpenStack Cisco ACI Deployment with OpFlex Plug-in



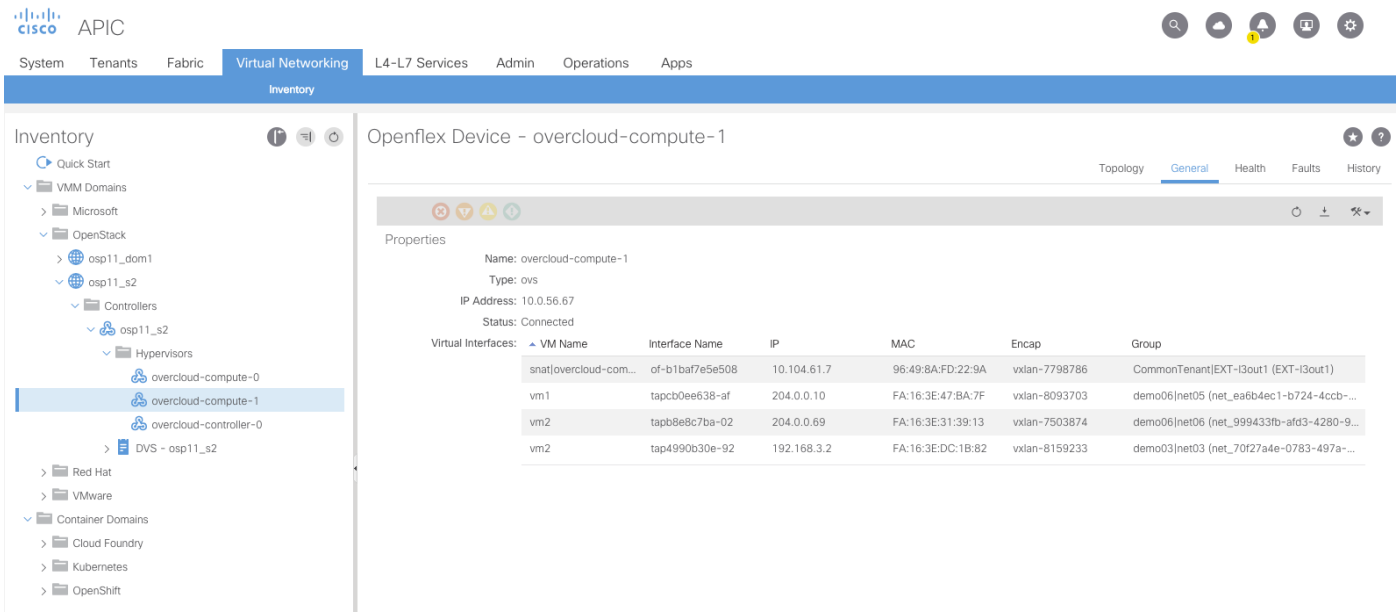
Note the following:

1. The Neutron datapath is replaced by the Cisco ACI fabric and the Plug-in for OpenStack. Consequently, there is no need for OpenStack Neutron nodes.
2. Cisco APIC provisions policies to the OpFlex agents running on the hosts using the Cisco ACI leaf switches as OpFlex P\proxies. These agents will in return provision OVS rules with OpenFlow.
3. The OpFlex agent creates an endpoint file containing VM name, IP address, network policy, and routing setting of all VMs. The Cisco ACI OpenStack plug-in uses this information to provide VM and virtual network information to Cisco APIC through the VMM domain.
4. Cisco ACI OpenStack integration also uses the OpFlex-Agent that runs OpFlex protocol with the Cisco ACI leaf and provisions the OVS rules through OpenFlow.
5. DHCP, metadata, NAT, policy enforcement, and forwarding are provided in a distributed fashion and are always enforced on the local compute node. The traffic between two VMs on the same compute node is routed locally by OVS. The traffic between two VMs on different compute nodes is routed always one hop away to the pervasive gateway on the leaf switch.
6. The encapsulation used between the compute node and the leaf switch can either be VXLAN or VLAN.

7. External connectivity is provided through Cisco ACI L3Out. This eliminates the requirement of Neutron or controller node being Layer 2 adjacent to the SNAT or floating IP network. The SNAT or floating IP networks are provided on the local OVS thanks to Cisco ACI policies.

8. When Cisco ACI plug-in runs in OpFlex mode, it configures a VMM domain for OpenStack and provides the information about hypervisors, VMs, IP or MAC address of VM, port groups, and encapsulation. Figure 15 shows a snapshot of Cisco APIC GUI OpenStack VMM domain view.

Figure 15: VMM Domain on APIC GUI



PhysDom Node Deployment

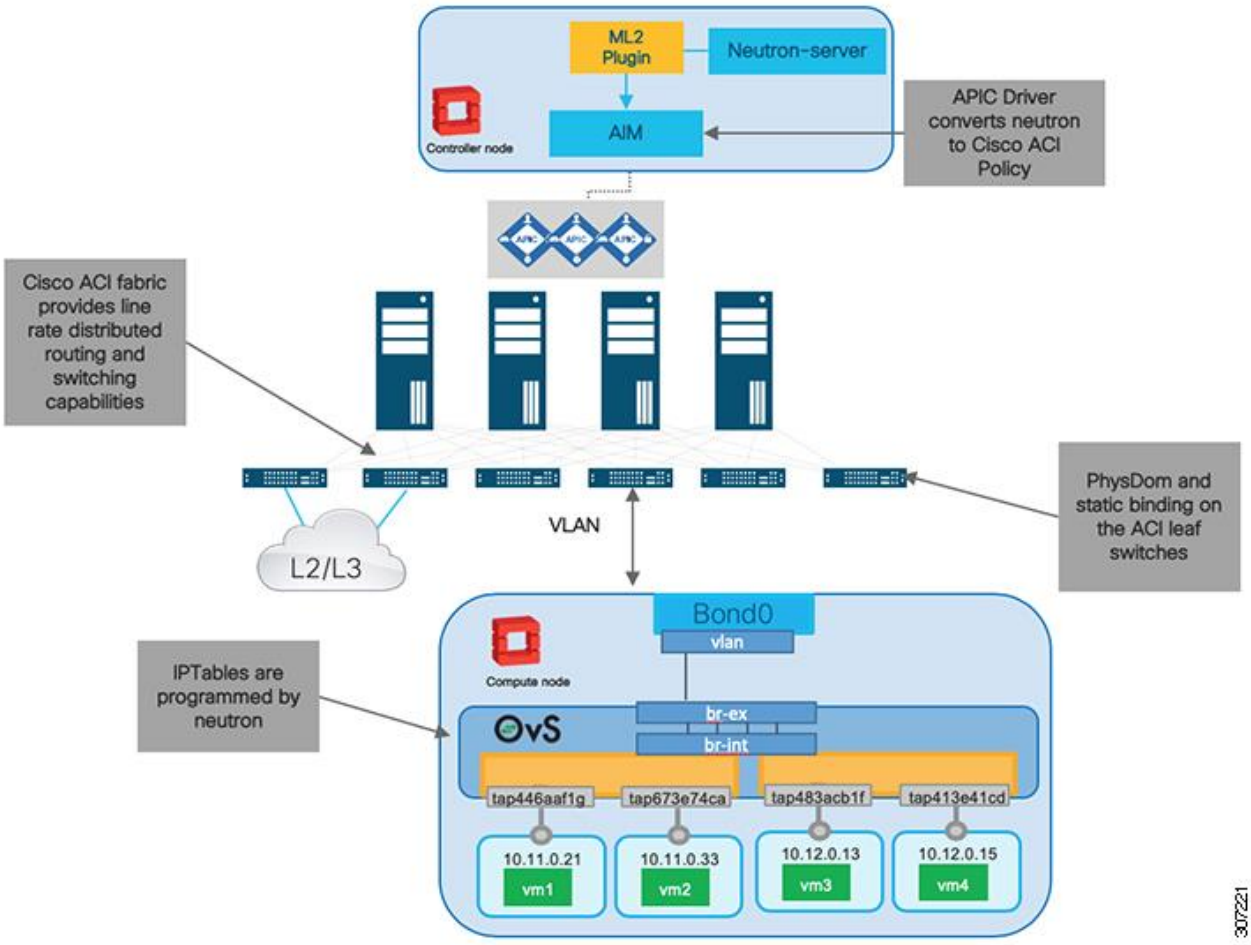
Cisco ACI Plug-in can run without OpFlex on the node, in which case we define the integration as PhysDom. As in the OpFlex model, here automatic configuration of EPGs, the bridge domain, and contracts is done on Cisco APIC as soon as Neutron networks are configured on OpenStack. However, compute nodes are statically bound to EPG with automated mapping of VLAN and leaf port.

For a deployment where none of the hosts has OpFlex agents, it is not mandatory to create a Cisco ACI VMM domain for OpenStack. Only a physical domain is used.

For deployments where a mix of OpFlex and PhysDom nodes are deployed, both physical and VMM Cisco ACI domains are mandatory.

Figure 16 illustrates the operation of the PhysDom Cisco ACI OpenStack operating model to highlights differences to the OpFlex one shown Figure 14.

Figure 16: OpenStack Cisco ACI Deployment with Non-OpFlex Plug-in



1. This model supports running standard OVS and accelerated datapath, for example, SR-IoV and OVS-DPDK technologies.
2. Neutron server directly deploys IpTable rules into the virtual switch. There is no intermediary as in OpFlex mode.
3. The OpenStack tenant networks can only use VLAN encapsulation between the compute node and the Cisco ACI leaf switch. The leaf in charge normalizes the VLAN-based flow into a VXLAN-based traffic.
4. You do not need to install any Cisco ACI-related component on the compute node. Also, if you use the standard OVS, it is provided by the OpenStack distribution installed.
5. NAT and SNAT are not supported in this mode.
6. Routing is enforced with the Cisco ACI pervasive gateway. The directly connected Cisco ACI leaf switch performs routing.
7. DHCP and metadata are not distributed services' OpenStack network node is required to provide these functions.

Optimized Routing, DHCP and Metadata Proxy Operations

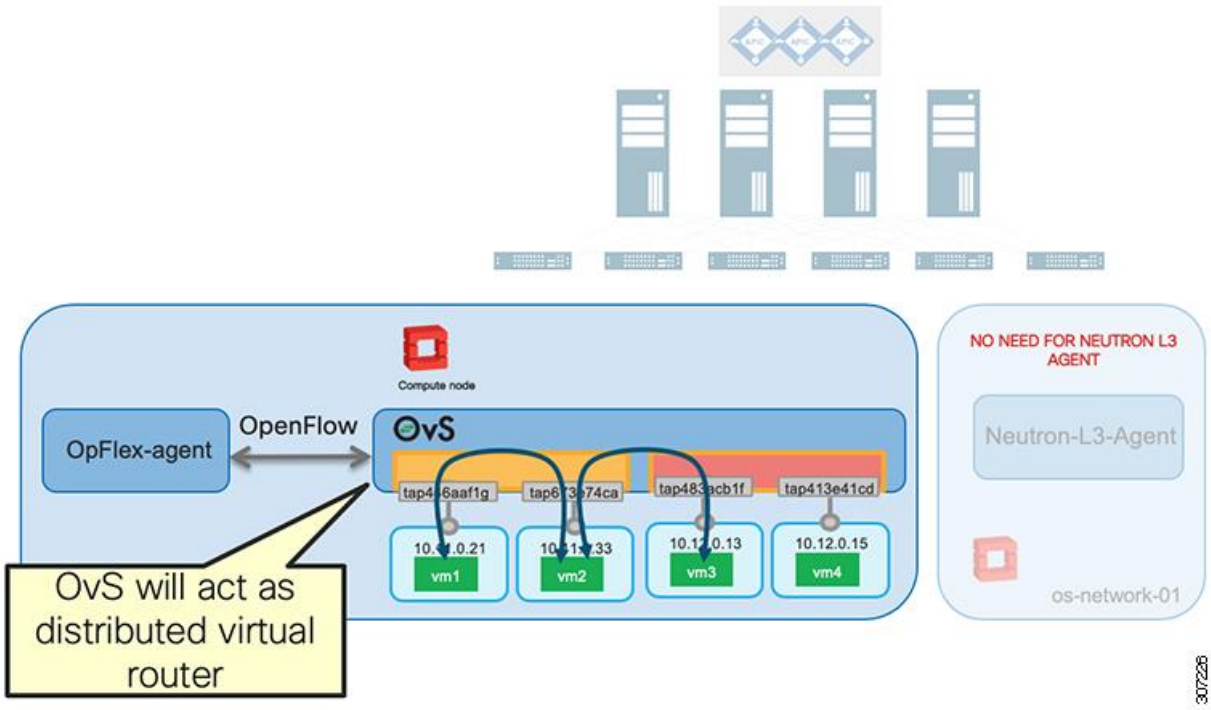
We have seen in the previous section that he unified ML2 driver software stack provides distributed routing and distributed processing to provide DHCP and metadata proxy services for VM instances. These services are designed to keep as much processing and packet traffic local to the compute host. The distributed elements communicate with centralized functions to ensure system consistency.

Distributed Routing Function

When you are running Cisco ACI unified plug-in for OpenStack in OpFlex mode, the Neutron data path is completely replaced. As such, the routing functions are distributed and enforced at the compute node level. Provided that the security groups are configured to allow the traffic, VMs running on OpenStack and bound to Neutron networks attached to the same Neutron router are routed directly by the local OVS running on the host. With this model, there is no need to trombone traffic in OpenStack network nodes performing the routing functions.

Figure 17 shows the routing flows for VMs running on the same hosts.

Figure 17: Distributed Routing

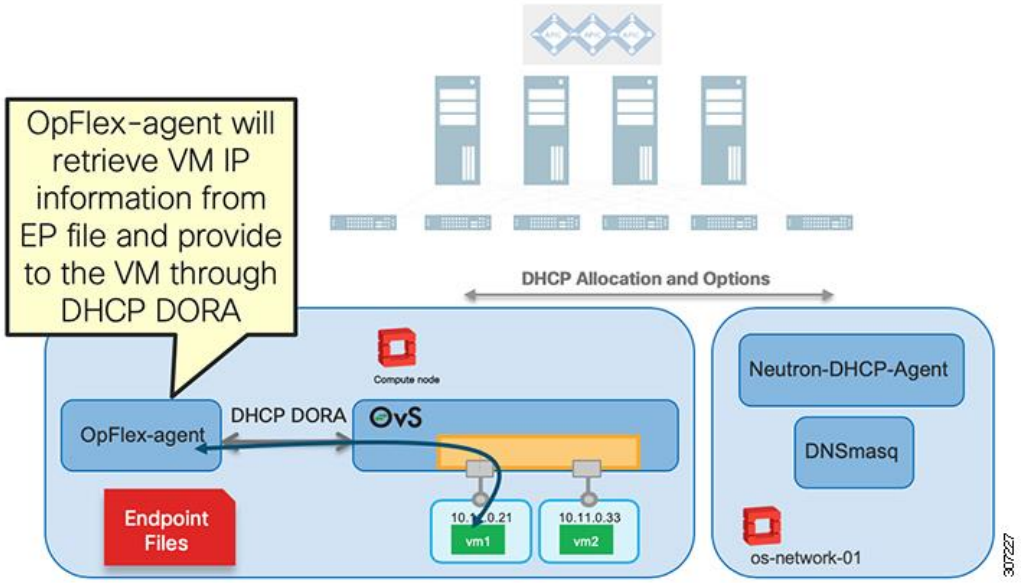


Neutron DHCP Optimization Service

OpenStack Neutron architecture uses the neutron-dhcp-agent service running on the Neutron server(s) to provide all DHCP communication to VM instances over the OpenStack tenant networks. The neutron-dhcp-agent provides central IP address management, and communicates with each VM instance for DHCP Discovery, Offer, Request, and Acknowledgement (DORA) functions.

The OpFlex-optimized DHCP approach instead provides all DORA services locally on the compute host through the OpFlex-Agent service. The distributed services communicate over the management network to the Neutron server(s) for allocation of IP addressing and DHCP options. This architecture keeps the bulk of the packet traffic required to issue a DHCP lease local to the compute host itself, while also offloading the processing of this interaction from the Neutron server(s). An illustration of this DHCP architecture is provided in Figure 18.

Figure 18: DHCP Optimization

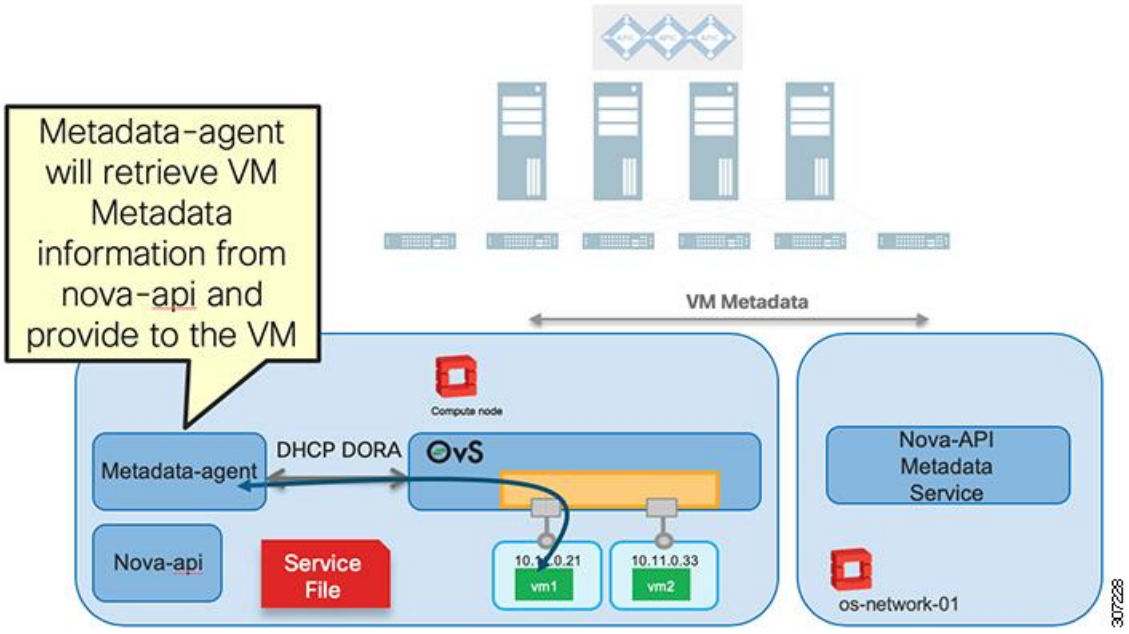


Neutron Metadata Optimization Service

OpenStack Neutron architecture for metadata delivery to VM instances provides a centralized proxy service running on the Neutron server(s). This proxy service looks up instance information in Nova API, adds HTTP headers, and redirects the metadata request to the Nova metadata service. The metadata requests from VM instances are transmitted on the OpenStack tenant networks.

The OpFlex-optimized metadata proxy approach instead provides metadata delivery using distributed metadata proxy instances running on each compute host. The OpFlex-Agent service reads the OpFlex service file and programs a flow in OVS to direct metadata service requests to the local neutron-metadata-agent. This local agent runs in a separate Linux namespace on the compute host. The metadata proxy function then accesses Nova-API and the Nova metadata service running on the OpenStack controller over the management network to deliver VM-specific metadata to each VM instance. An illustration of this metadata proxy architecture is shown in Figure 19.

Figure 19: Metadata Agent Flow



Support for Neutron Networks

Table 7 summarizes the supported Neutron networks with Cisco ACI Plug-in for OpenStack.

Table 7: Supported Neutron Networks

router-external	shared	network type	Cisco ACI support	Type SVI	Description
False	False	OpFlex/VLAN	Yes	No	Typical tenant network. Only usable by members of the tenant. VMs can be plugged here only from tenant owner.
False	True	OpFlexVLAN	Yes	No	This can be shared by multiple tenants with RBAC on who can use it. VMs can be plugged here from multiple tenants.

True	False	OpFlex	Yes	No	External network. Tenants can optionally set gateway for their OpenStack router to one external network.
True	True	OpFlex	Yes	No	External network. Scope is all tenants. Can only be created by administrators. Tenants can optionally set a gateway for their OpenStack router to one external network.
True	f False	VLAN	Yes	Yes	This is a special tenant network. VMs running on OpenStack can connect here. This is represented on Cisco ACI as an L3Out. Typically, VMs attached to this network type are VNFs capable of doing dynamic routing protocol (BGP) with Cisco ACI L3Out and advertising VIP services.

External Neutron Networks

Cisco ACI Plug-in for OpenStack supports the creation of external Neutron networks.

External networks can be defined either shared or dedicated:

- A shared external network is visible by all OpenStack projects.
- A dedicated connectivity for the OpenStack project.

It is possible to have a mixed environment with shared and dedicated external connectivity.

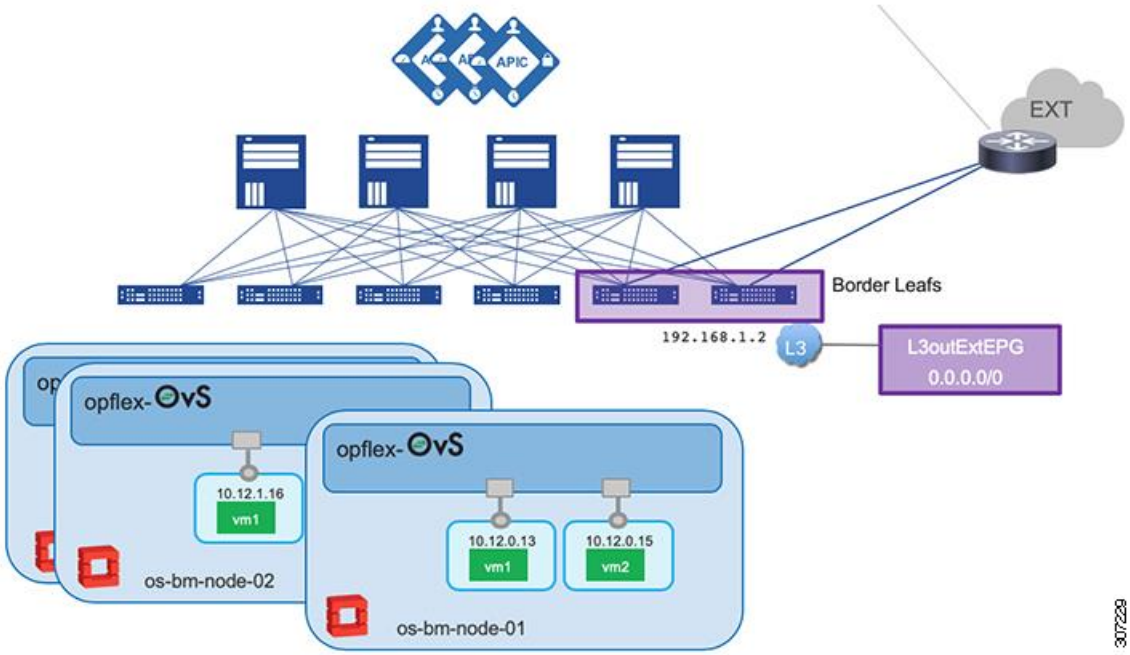
Cisco ACI L3Out must be defined on Cisco ACI directly by any means, such as REST API, CLI, or Cisco APIC GUI. The Cisco ACI L3Out can be configured for static or dynamic routing protocol with external routers. How the Cisco ACI l3Out is configured is completely transparent for the OpenStack administrator. OpenStack Neutron simply consumes an existing Cisco ACI L3Out resource by knowing the L3Out and related external EPG distinguished names. This is shown in the following CLI example where the distinguished name for the L3Out and L3Out external EPG is uni/tn-prj_$demo01/out-l3out1/instP-extEpg:

```
[stack@dom-undercloud02 demo01_DNAT_dedicated]$ neutron net-create external-net-dedicated --
router:external --apic:distinguished_names type=dict ExternalNetwork=uni/tn-prj_$demo01/out-
l3out1/instP-extEpg

Created a new network:
```

```
+---------------------------+------------------------------------------------------------------+
| admin_state_up            | True                                                             |
| apic:bgp_asn              | 0                                                                |
| apic:bgp_enable           | False                                                            |
| apic:bgp_type             | default_export                                                   |
| apic:distinguished_names  | {"ExternalNetwork": "uni/tn-prj_53c5620850904dbf882477a679641|
|                           | 6af/out-l3out1/instP-extEpg", "BridgeDomain": "uni/tn-prj_53c|
|                           | 5620850904dbf882477a6796416af/BD-EXT-l3out1", "VRF": "uni/tn-|
|                           | prj_53c5620850904dbf882477a6796416af/ctx-externalVRF", "Endpo|
|                           | intGroup": "uni/tn-prj_53c5620850904dbf882477a6796416af/ap-Op|
|                           | enStack/epg-EXT-l3out1"}                                         |
| apic:external_cidrs       | 0.0.0.0/0                                                        |
| apic:nat_type             | distributed                                                      |
| apic:svi                  | False                                                            |
| apic:synchronization_state | synced                                                          |
| created_at                | 2018-12-06T14:37:39Z                                             |
| id                        | 73baddea-e70b-4a81-8b81-acf730f54377                             |
| mtu                       | 1500                                                             |
| name                      | external-net-dedicated                                           |
| port_security_enabled     | True                                                             |
| project_id                | 53c5620850904dbf882477a6796416af                                 |
| provider:network_type     | opflex                                                           |
| provider:physical_network | physnet1                                                         |
| revision_number           | 4                                                                |
| router:external           | True                                                             |
| shared                    | False                                                            |
| status                    | ACTIVE                                                           |

| subnets                   |                                                                  |
```

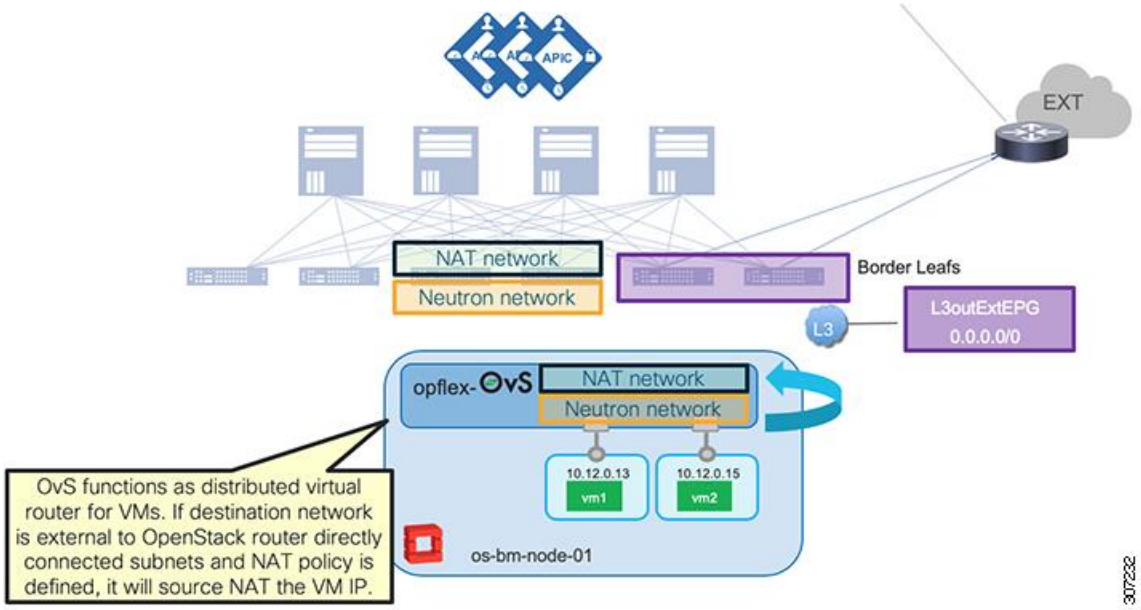
```
| tenant_id                   | 53c5620850904dbf882477a6796416af                                 |
+-----------------------------+------------------------------------------------------------------+
```

Figure 20: L3Out Layout



When NAT is enabled, OVS applies distributed NAT policies to assign source NAT or FIP address to VM traffic. The OVS automatically applies NAT rules when the destination of the traffic is not directly connected to the OpenStack router that the VM subnet belongs to.

Figure 21: OVS Functions as Distributed Virtual Router



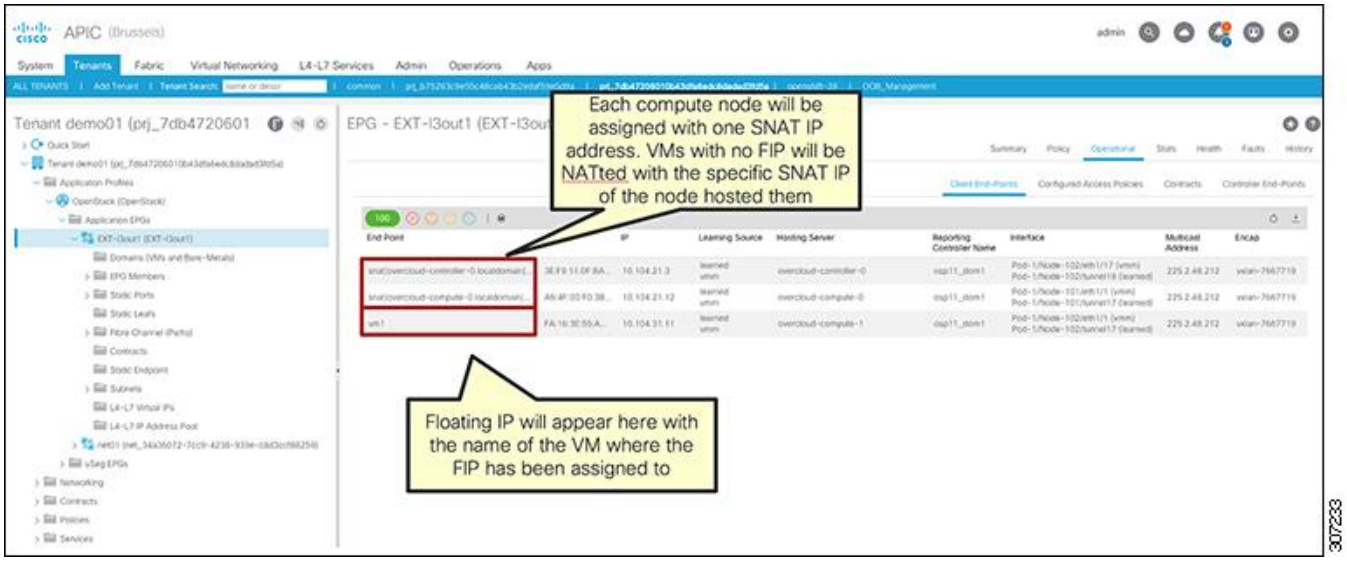
When OVS translates the packet address, the packet also placed into a special Cisco ACI EPG segment called external EPG. Do not confuse this external EPG with the L3Out external EPG. It is a standard EPG that exposes all the SNAT IP addresses as well the floating IP addresses assigned to OpenStack. The external EPG also has its own encapsulation that is used by the OVS to tag traffic towards the Cisco ACI leaf switch.

When SNAT is used, each compute node is automatically assigned with one IP address, which is used to NAT VMs hosted in that server.

Figure 22: SNAT and FIP IP Addresses Shown in the Cisco APIC GUI

The NAT subnet is attached to the external Neutron network and marked through the CLI keyword apic:snat_host_pool True as shown in the following CLI command example:

```
[stack@dom-undercloud02    demo01_DNAT_dedicated]$ neutron subnet-create external-net-dedicated
10.104.21.0/24 --name ext-subnet --disable-dhcp --gateway 10.104.21.1 --apic:snat_host_pool True
```




```
Created a new subnet:
| allocation_pools           | {"start": "10.104.21.2", "end": "10.104.21.254"} |
| apic:distinguished_names   | {}                                               |
| apic:snat_host_pool        | True                                             |
| apic:synchronization_state | N/A                                              |
| cidr                       | 10.104.21.0/24                                   |
| created_at                 | 2018-12-06T14:37:53Z                             |
| description                |                                                  |
| dns_nameservers            |                                                  |
| enable_dhcp                | False                                            |
| gateway_ip                 | 10.104.21.1                                      |
| host_routes                |                                                  |
| id                         | 6620b166-b49c-4153-80a8-07b53709738a             |
| ip_version                 | 4                                                |
```

```
| ipv6_address_mode         |                                                    |
| ipv6_ra_mode              |                                                    |
| name                      | ext-subnet                                         |
| network_id                | 73baddea-e70b-4a81-8b81-acf730f54377               |
| project_id                | 53c5620850904dbf882477a6796416af                   |
| revision_number           | 2                                                  |
| service_types             |                                                    |
| subnetpool_id             |                                                    |
| tags                      |                                                    |
| tenant_id                 | 53c5620850904dbf882477a6796416af                   |
+---------------------------+----------------------------------------------------+
```

Dedicated External Network

A dedicated external network can be defined to provide external connectivity to specific tenants. This type of connectivity is typically provided for OpenStack projects that require dedicated L3Out connectivity. In this scenario, a dedicated L3Out must be defined in the OpenStack-created user Cisco ACI tenant. ny tool can be used to create this L3Out, such as REST API, CLI, or the Cisco APIC GUI.

The L3Out defines how Cisco ACI will connect to external routers and will specify such information as:

- Interfaces and their IP information
- Dynamic routing, if any
- External EPG

You should *not* add any contract because contracts are added automatically by the Cisco ACI Plug-in for OpenStack.

Note that if you require SNAT or FIP enabled for this external network, the L3Out must be defined in a different VRF from the one created by the Cisco ACI Plug-in for OpenStack. If you do not require SNAT or FIP to be enabled, the L3Out must be defined instead on the same VRF created in the Cisco ACI tenant by the Plug-in. The VRF created by OpenStack on Cisco ACI is called *DeafultVRF*.

Shared External Network

A shared external network can be defined to provide external connectivity to a subset of OpenStack projects. This type of connectivity is typically provided when there is limited L3Out connectivity to the external world, or simply when it is not a concern to share the same L3Out with multiple projects.

In this case, an L3Out must be defined in the common Cisco ACI tenant. Any tool can be used to create this L3Out.

The L3Out defines how Cisco ACI connects to external routers and specifies such information as:

- Interfaces and their IP information

- Dynamic routing if any
- External EPG

Do *not* add any contract. Contracts are added later automatically by the Cisco ACI Plug-in for OpenStack.

Note that SNAT or FIP must be enabled for shared external network because the L3Out and the OpenStack project are by default defined in different VRFs.

As said before, when you do not want NAT enabled, the Neutron network and L3Out in the common tenant must be attached to the same VRF. For this use case, you must use address scope and define the OpenStack subnet in the same VRF as the shared L3Out.

The following example shows how to create an address scope pointing to the distinguied name of the L3Out VRF, in this case being uni/tn-common/ctx-VRFT1T2:

```
[stack@dom-undercloud02 demo05_address_scope]$ neutron address-scope-create --shared VRFT1T2 4 --apic:distinguished_names type=dict VRF=uni/tn-common/ctx-VRFT1T2

Created a new address_scope:

| apic:distinguished_names   | {"VRF": "uni/tn-common/ctx-VRFT1T2"} |

| apic:synchronization_state | synced                               |

| id                         | 3a235d69-4fc3-4d2a-893e-b82e6d33b16d |

| ip_version                 | 4                                    |

| name                       | VRFT1T2                              |

| project_id                 | 917eaef7324844698642eb24738b5ca8     |

| shared                     | True                                 |

| tenant_id                  | 917eaef7324844698642eb24738b5ca8     |

+----------------------------+--------------------------------------+
```

Once the address scope is created, you add a subnet pool to this address scope, as shown in the following example:

```
[stack@dom-undercloud02 demo05_address_scope]$ neutron subnetpool-create --address-scope VRFT1T2
--shared --pool-prefix 204.0.0.0/21 --default-prefixlen 26 subnetpool

Created a new subnetpool:

| address_scope_id  | 3a235d69-4fc3-4d2a-893e-b82e6d33b16d |

| created_at        | 2018-12-06T15:16:03Z                 |

| default_prefixlen | 26                                   |

| id                | 5bf15775-30eb-4f08-a771-6106e75243d7 |

| ip_version        | 4                                    |

| is_default        | False                                |

| is_implicit       | False                                |

| max_prefixlen     | 32                                   |

| min_prefixlen     | 8                                    |

| name              | subnetpool                           |

| prefixes          | 204.0.0.0/21                         |

| project_id        | 917eaef7324844698642eb24738b5ca8     |

| shared            | True                                 |

+-------------------+--------------------------------------+
```

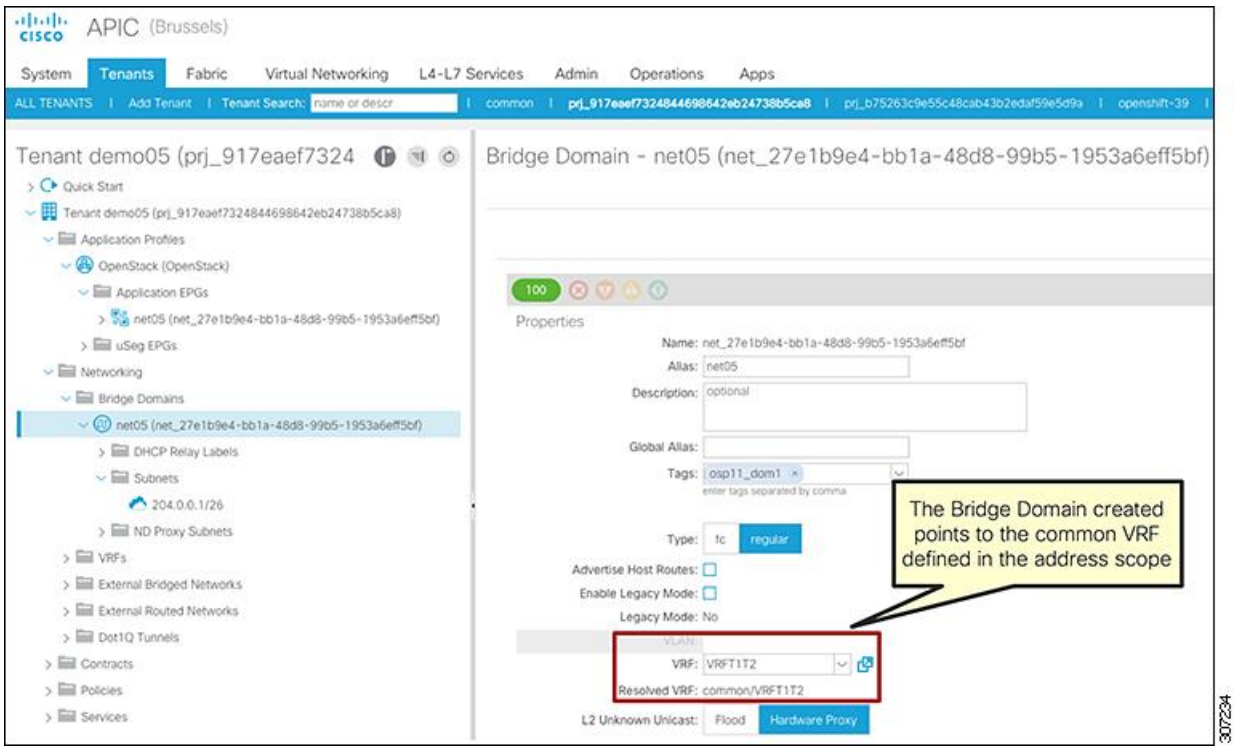
Finally, you add network and subnet, pointing the latter to the subnet pool created, as shown in the following example:

```
[stack@dom-undercloud02 demo05_address_scope]$ neutron net-create net05

[...]

[stack@dom-undercloud02 demo05_address_scope]$ neutron subnet-create --name subnet05 --
subnetpool net05

neutron CLI is deprecated and will be removed in the future. Use OpenStack CLI instead.

Created a new subnet:

| allocation_pools           | {"start": "204.0.0.2", "end": "204.0.0.62"} |

| apic:distinguished_names   | {}                                          |

| apic:snat_host_pool        | False                                       |
```

```
| apic:synchronization_state | N/A                                      |
| cidr                       | 204.0.0.0/26                             |
| created_at                 | 2018-12-06T15:16:08Z                     |
| dns_nameservers            |                                          |
| enable_dhcp                | True                                     |
| gateway_ip                 | 204.0.0.1                                |
| host_routes                |                                          |
| id                         | 257c9457-f90f-4a63-b06f-2f2effb463c7     |
| ip_version                 | 4                                        |
| ipv6_address_mode          |                                          |
| ipv6_ra_mode               |                                          |
| name                       | subnet05                                 |
| network_id                 | 27e1b9e4-bb1a-48d8-99b5-1953a6eff5bf     |
| project_id                 | 917eaef7324844698642eb24738b5ca8         |
| revision_number            | 2                                        |
| subnetpool_id              | 5bf15775-30eb-4f08-a771-6106e75243d7     |
+----------------------------+------------------------------------------+
```

On Cisco ACI, the bridge domain points to the VRF as defined in the address scope as shown in Figure 23.

Figure 23: Address Scope



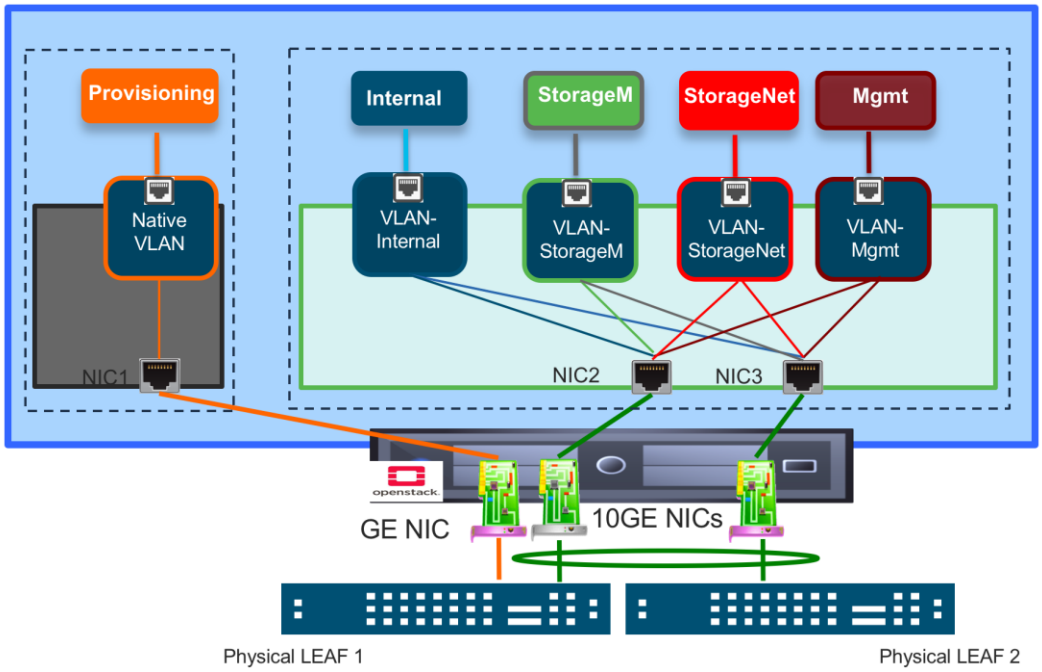
Infrastructure Architecture

Cisco ACI and OpenStack Physical Architecture

A typical architecture for a Cisco ACI fabric with an OpenStack deployment consists of" a Nexus 9000 Series spine and leaf topology; a Cisco APIC cluster; a group of servers to run the various provisioning, control, storage and compute components of OpenStack; and routers providing external network connectivity.

OpenStack nodes are connected to Cisco ACI leaf switches. The number of NICs used for each server depend on bandwidth and high availability requirements of the applications.

Figure 24 shows a typical way to connect the OpenStack nodes to the Cisco ACI leaf switches.



One interface, typically a gigabit interface, is dedicated to provisioning capability, such as PXE boot, and therefore requires native VLAN connectivity.

At least two other interfaces for redundancy, typically 10 Gigabit capable, are dedicated to connectivity of various networks, such as internal, storage management, storage network, and external. Tenant connectivity also uses these two interfaces.

Bonding can also be used on the host and VPC on the leaf side. VPC configurations don't require a peer link with Cisco ACI: bonding offers a simpler and lower-cost redundancy on the leaf side.

In highly scalable OpenStack environments, it is a best practice to connect the OpenStack controller to dedicated leaf pairs. The leaf switches act as an OpFlex proxy. Because controllers join all networks to provide DHCP capabilities, they require more resources from the OpFlex proxy leaf switches they are connected to compared with any other compute node.

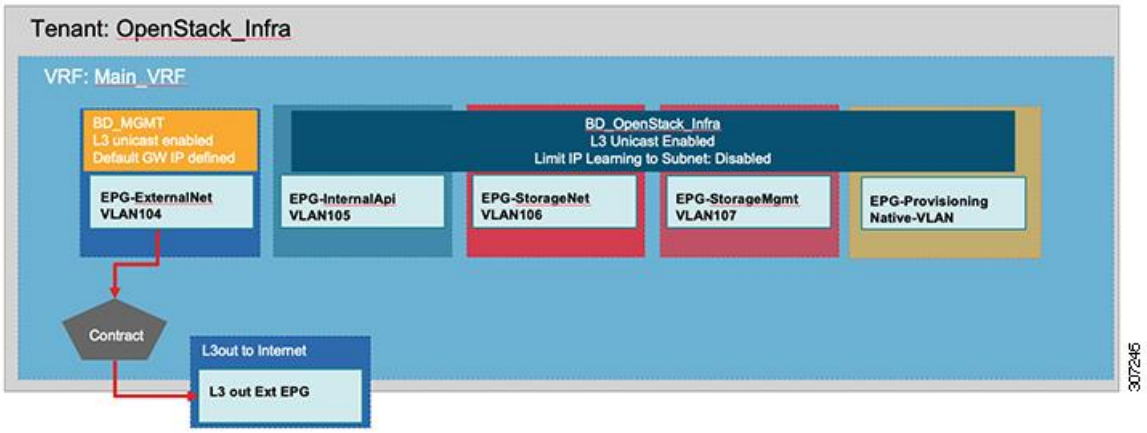
OpenStack tenant and VM connectivity are provisioned automatically through the Cisco ACI plug-in for OpenStack.

OpenStack infrastructure connectivity across nodes must be preprovisioned on the Cisco ACI fabric to allow the OpenStack nodes to communicate with each other. Typically, there are several networks required for OpenStack:

- Internal API Network (VLAN)
- Storage Network (VLAN)
- Storage Management Network (VLAN)
- Provisioning Network (Native VLAN)
- External Network (VLAN)

A Cisco ACI tenant can be dedicated to provide connectivity. Figure 25 shows an example of how a dedicated tenant can be configured to provide connectivity.

Figure 25: Cisco ACI Tenant for OpenStack Management Networks



In this example, two bridge domains are created:

- BD_MGMT provides OOB connectivity. (In this example this provides connectivity both Internet and the Cisco APIC controller.)
- BD_OSP does not require any gateway because it serves only isolated networks. However, as a best practice, the example keeps Layer 3 enabled to learn the IP address from the hosts and have a better visibility of the nodes.

The EPGs have static bindings to the interfaces of the connected hosts and use a physical domain.

Note that the EPG ExternalNet generally requires Internet connectivity because it used by the OpenStack nodes to run updates and download packages. In this example, the L3Oout providing connectivity to Internet and the Cisco APIC management OOB network is defined in the same Cisco ACI tenant. However, it could use an L3Out defined in the common tenant.

It is not necessary to use a dedicated Cisco ACI tenant for infrastructure connectivity. The Cisco ACI administrator could reuse any other existing tenant (including the common tenant) to provision this configuration.

Although not a strict requirement, it is usually a best practice to define a dedicated Cisco ACI tenant for all infra connectivity of the different hypervisors, including OpenStack. This simplifies troubleshooting.

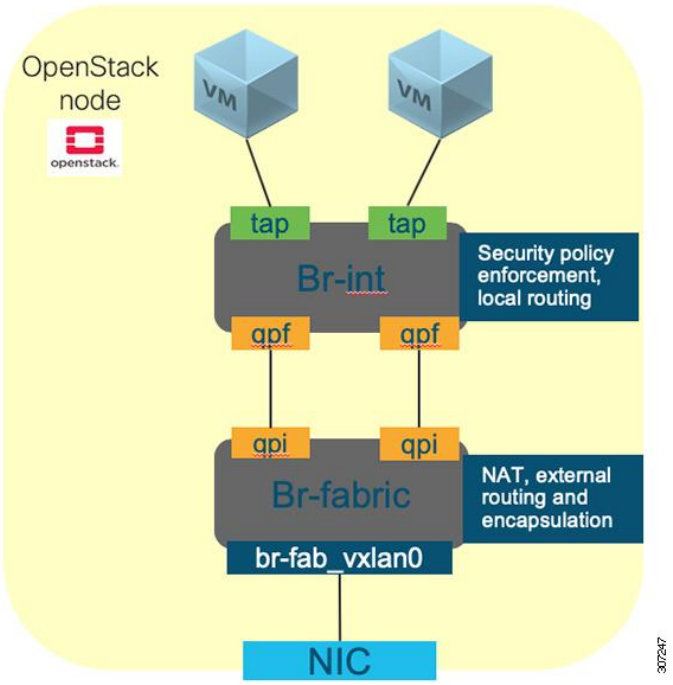
Life of a Packet with Open vSwitch and OpFlex Cisco ACI Plug-in

When the OpFlex and OVS agents are installed on the compute nodes, the Neutron datapath is replaced by the Cisco ACI fabric and Cisco ACI Plug-in for OpenStack.

Figure 26 shows the typical view of the OVS bridges existing on a OpenStack compute node. While the integration bridge performs the routing local to the host and security policy enforcement, the fabric bridge is intended for more advanced feature capabilities, such as NAT and encapsulation to the uplink interface of the host.

Having multiple bridges performing different functions help to improve performance and optimize traffic throughput.

Figure 26: OpenStack Nodes Integration Bridges



The Endpoint File

The Neutron OpFlex Agent populates the endpoint files of the compute nodes located in the /var/lib/OpFlex-agent-ovs/endpoints/ directory. One file is created for each instance running on the host. The file contains the metadata describing the network capabilities of the instance. The following is an example of the endpoint file.

```
{
    "neutron-metadata-optimization": true,
    "dhcp4": {
        "domain": "openstacklocal",
        "static-routes": [
            {
                "dest": "0.0.0.0",
                "next-hop": "192.168.1.254",
                "dest-prefix": 0
            },
            {
```

```
                "dest": "169.254.0.0",
                "next-hop": "192.168.1.1",
                "dest-prefix": 16
            }
        ],
        "ip": "192.168.1.11",
        "interface-mtu": 1500,
        "prefix-len": 24,
        "server-mac": "fa:16:3e:3f:ba:cc",
        "routers": [
            "192.168.1.254"
        ],
        "server-ip": "192.168.1.1",
        "dns-servers": [
            "192.168.1.1"
        ]
    },
    "access-interface": "tap1fa675c5-c6",
"ip": [
        "192.168.1.11"
    ],
    "promiscuous-mode": false,
    "anycast-return-ip": [
        "192.168.1.11"
    ],
    "mac": "fa:16:3e:68:36:e4",
    "domain-name": "DefaultVRF",
    "domain-policy-space": "prj_7d0be879a12c47ae9c0a26d3fd4407d1",
```

```
"ip-address-mapping": [
    {
        "mapped-ip": "192.168.1.11",
        "next-hop-mac": "96:49:8a:fd:22:9a",
        "uuid": "9190dd57-012e-48a0-ac0a-05d191812bca",
        "floating-ip": "169.254.0.2",
        "next-hop-if": "of-b1baf7e5e508",
        "policy-space-name": "common",
        "endpoint-group-name": "osp11_s2_OpenStack|EXT-l3out1"
    }
],
"endpoint-group-name": "OpenStack|net_016b9885-c8ac-4a2d-be7e-e5203c945ba4",
"access-uplink-interface": "qpf1fa675c5-c6",
"neutron-network": "016b9885-c8ac-4a2d-be7e-e5203c945ba4",
"interface-name": "qpi1fa675c5-c6",
"security-group": [
    {
        "policy-space": "prj_7d0be879a12c47ae9c0a26d3fd4407d1",
        "name": "6913480e-5bf5-4903-b2b1-a394dd3242ea"
    },
    {
        "policy-space": "common",
        "name": "osp11_s2_DefaultSecurityGroup"
    }
],
"uuid": "1fa675c5-c67f-41ee-a855-3389612e1768|fa-16-3e-68-36-e4",
"policy-space-name": "prj_7d0be879a12c47ae9c0a26d3fd4407d1",
"attributes": {
```

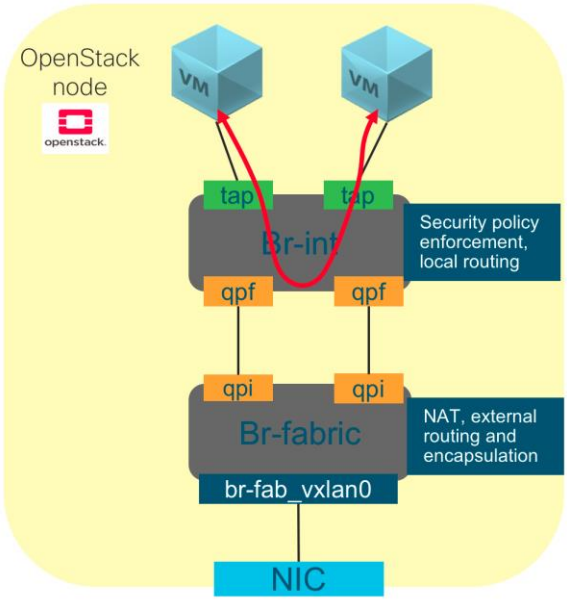
```
        "vm-name": "vm3"
    }
}
```

The information contained in the endpoint file are used by the OpFlex-Agent to configure Open vSwitch rules.

Traffic Between Instances on the Same Host

Traffic between two VMs running on the same hypervisor hit the internal bridge, which applies security enforcement rules. If the security group allows, the OVS will switch or route the packet to the destination endpoint.

Figure 27: Traffic on the Same Host



Traffic Between Instances on Different Hosts

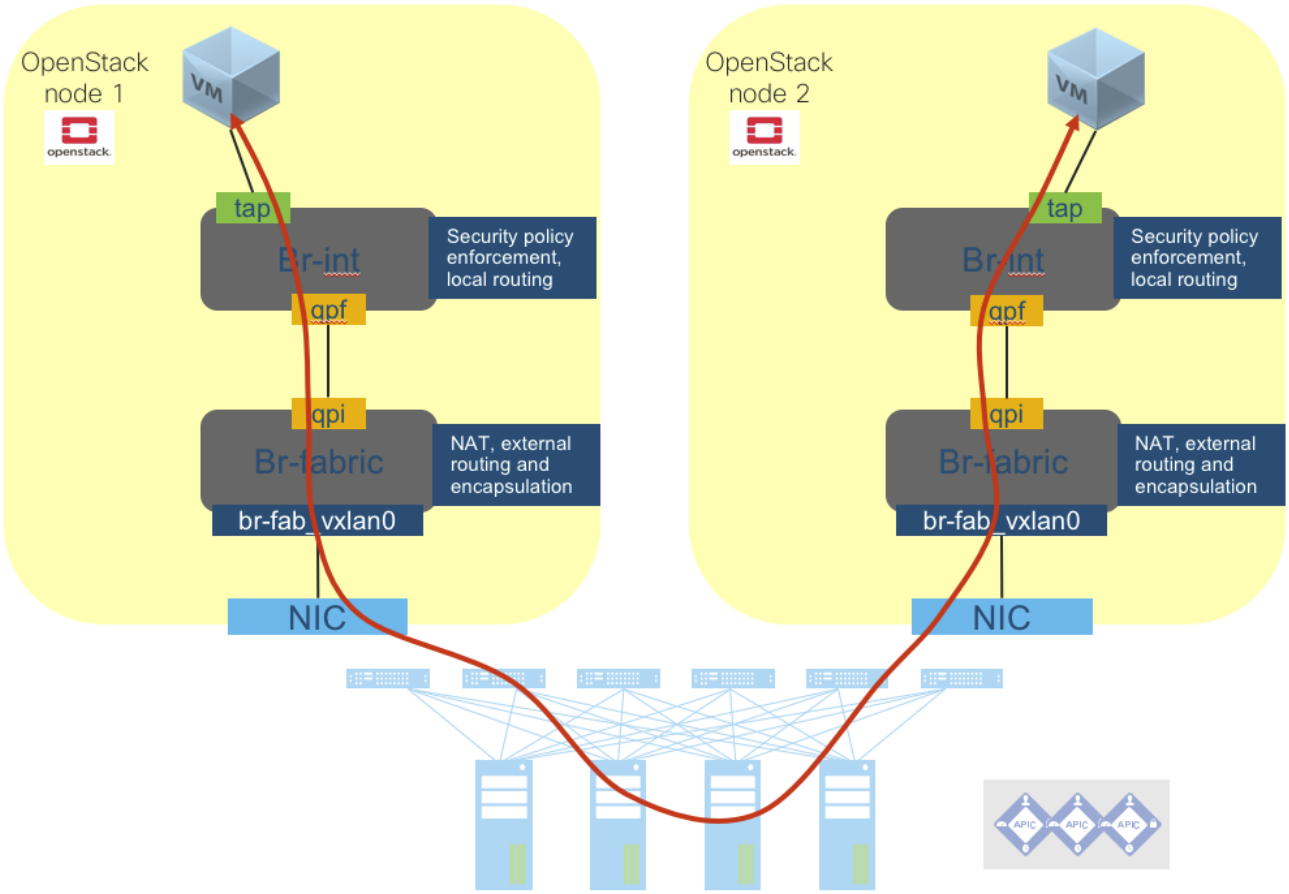
When the destination of a packet is another OpenStack VM belonging to a different compute node than the one hosting the source instance, traffic crosses both integration and fabric bridges.

Note that this scenario is referring to OpenStack VMs belonging to the same OpenStack project.

The integration bridges allow or deny communication based on the security groups.

Fabric bridges take care of encapsulating and routing the packets to the physical NICs towards the Cisco ACI fabric.

Figure 28: Traffic Between VMs in Different Hosts



Cisco ACI leaf switches also verify policy enforcement (based on Cisco ACI contracts) and route the packets towards the compute node hosting the destination VM.

Note: In the current Cisco ACI OpFlex implementation with VXLAN encapsulation, in the scenario where the destination VM is in the same subnet as the source VM, the source compute node encapsulates the unicast traffic into multicast using the EPG multicast destination for the packet. For this reason, only VLAN is supported with blade servers.

For troubleshooting purposes, to sniff traffic on br-int and br-fab, it is necessary to run the following commands:

```
ip link add name br-int-snooper0 type dummy

ip link set dev br-int-snooper0 up

ovs-vsctl -- set Bridge br-int mirrors=@m  -- --id=@br-int-snooper0 get Port br-int-snooper0
-- --id=@br-int get Port br-int -- --id=@m create Mirror name=mymirror1 select-dst-port=@br-
int select-src-port=@br-int output-port=@br-int-snooper0 select_all=1

tcpdump -eni br-int-snooper0

ip link add name br-fab-snooper0 type dummy
```

```
ip link set dev br-fab-snooper0 up

ovs-vsctl add-port br-fabric br-fab-snooper0

ovs-vsctl -- set Bridge br-fabric mirrors=@m  -- --id=@br-fab-snooper0 get Port br-fab-
snooper0 -- --id=@br-fabric get Port br-fabric -- --id=@m create Mirror name=mymirror1
select-dst-port=@br-fabric select-src-port=@br-fabric output-port=@br-fab-snooper0
select_all=1

tcpdump -eni br-fab-snooper0
```

Traffic Between an OpenStack Instance and an External Subnet

When the destination of a packet is a subnet not belonging to the OpenStack project where the source VM is hosted, Open vSwitch treats this as a packet destined to external router. Traffic crosses both integration and fabric bridges of the OpenStack compute node hosting the source VM and reaches the Cisco ACI leaf switch.

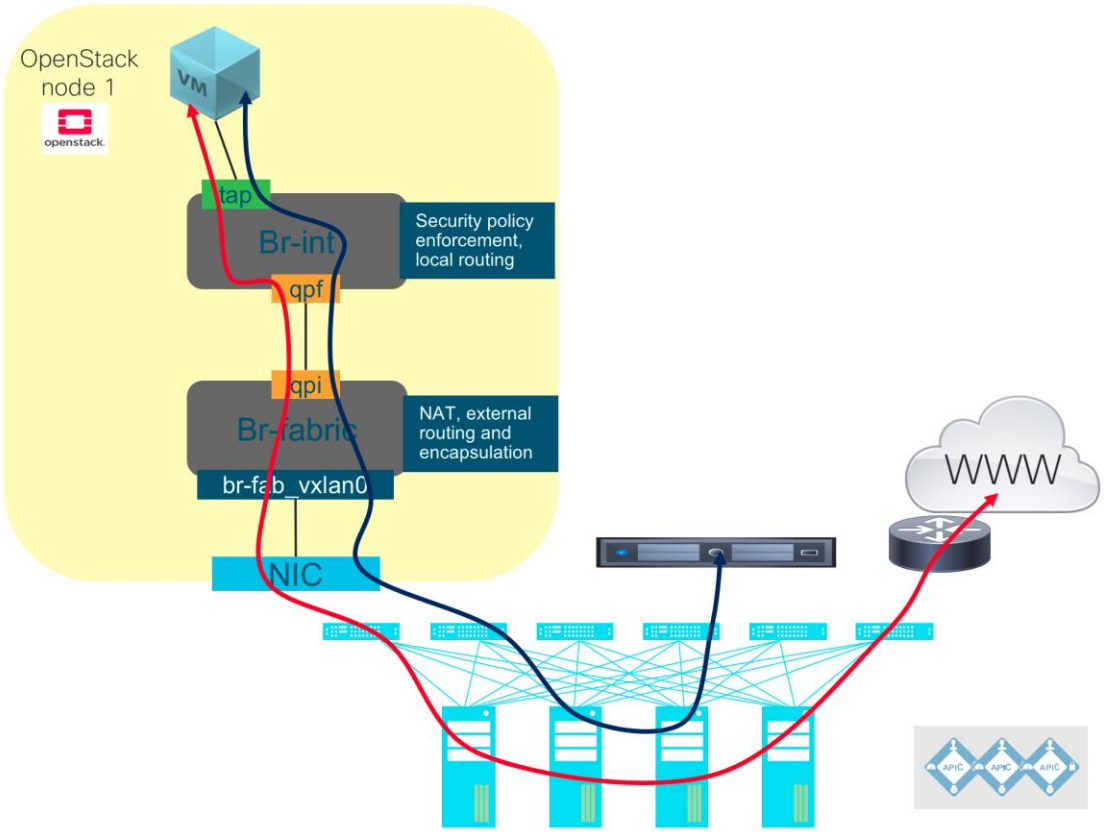
The integration bridge allows or denies communication based on the security groups applied to the source VM and destination subnet.

The fabric bridge will encapsulate and route the packets to the physical NICs towards the Cisco ACI fabric. The fabric bridge also applies SNAT or floating IP rules if required.

Once the packet reaches the Cisco ACI leaf, Cisco ACI determines if the destination is an endpoint directly connected to the fabric, or if it is a subnet behind an L3Out facing, for example, an external router. If Cisco ACI contracts allow, the fabric routes the packet to the destination as shown in Figure 29.

.

Figure 29: Traffic to the External OpenStack Subnet



Appendix

This appendix provides a brief summary of the Cisco ACI Plug-in for OpenStack constructs and resources.

OpenStack plug-in Constructs

APIC System ID

A unique identifier for the OpenStack instance. Most of the fabric access policies resources created by the plug-in use this as a prefix. This ID needs to be unique for each OpenStack cluster for each Cisco APIC.

Encap Mode

The encapsulation used for communication between the OpenStack nodes and the leaf switches. This can be either VLAN or VXLAN. This is independent of the tenant network type, which in most cases will be OpFlex or VLAN.

OpFlex Network Type

A new type of virtual network introduced and used by Cisco ACI OpenStack Plug-in. This is an abstract network type that uses a Cisco ACI VXLAN overlay fabric within Cisco ACI. It can be used with different encapsulation modes (VLAN or VXLAN) when entering or leaving the Cisco ACI fabric.

SNAT Pool

Because the Cisco ACI OpenStack plug-in uses distributed NAT, each compute node is responsible for performing NAT on all the instances running on the compute node. Each compute node gets one IP address (or sometimes multiple IP addresses depending on configuration) to use for SNAT for instances that do not have a floating IP address.

Floating IP Pool

The standard neutron floating IP pool/subnet.

OpenStack Instance

Each OpenStack instance is identified by the unique identifier called apic_system_id. A VMM domain is usually created using the same identifier, and any common assets created in Cisco APIC are also prefixed with it.

Project

Each project created in OpenStack is represented as a Cisco ACI tenant and an application profile within that tenant.

Tenant Network

A tenant network is represented as an EPG and bridge domain pair created within the Cisco ACI tenant corresponding to the project.

Subnet

When a subnet is added to a Neutron network, it appears as a subnet added to the bridge domain corresponding to the Neutron network. Keep in mind that the subnet only appears in the bridge domain if the tenant network is attached to a tenant router.

Router

A router in Neutron is represented as a contract applied between the EPGs that correspond to the tenant networks that the router is attached to.

External Network

External or provider networks are provided using L3Out. The external network is created similarly to other mechanisms using the neutron net-create command. The distinguished name of the L3Out-EPG is passed as a parameter to tell the plug-in which L3Out to use for this external network. After that, at least two subnets are added to the network, one for the SNAT pool and other for the floating IP pool.