Quantum Documentation

Release 2012.2.4-dev

Quantum development team

July 30, 2013

CONTENTS

Quantum is an OpenStack project to provide "network connectivity as a service" between interface devices (e.g., vNICs) managed by other Openstack services (e.g., nova). It implements the Quantum API Guide.

This document describes Quantum for contributors of the project, and assumes that you are already familiar with Quantum from an end-user perspective.

This documentation is generated by the Sphinx toolkit and lives in the source tree. Additional documentation on Quantum and other components of OpenStack can be found on the OpenStack wiki. The Quantum Development wiki is a very good place to start.

Enjoy!

CHAPTER

ONE

DEVELOPMENT DOCUMENTS

1.1 Developer Guide

The Quantum Wiki is a very good place to start.

1.2 Open Stack Common

A number of modules used are from the openstack-common project. The imported files are in 'quantum/openstack-common.conf'. More information can be found at OpenStack Common.

1.3 Quantum Sources (modified by POL)

1.3.1 Source: quantum/agent/linux/interface.py

class quantum.agent.linux.interface.**BridgeInterfaceDriver**(*conf*)
Bases: quantum.agent.linux.interface.LinuxInterfaceDriver

Driver for creating bridge interfaces.

DEV_NAME_PREFIX = 'ns-'

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)
Plugin the interface.

unplug(*device_name*, *bridge=None*, *namespace=None*, *prefix=None*)
Unplug the interface.

class quantum.agent.linux.interface.**LinuxInterfaceDriver**(*conf*)
Bases: object

DEV_NAME_LEN = 14

DEV_NAME_PREFIX = 'tap'

check_bridge_exists(*bridge*)

get_device_name(*port*)

init_l3(*device_name*, *ip_cidrs*, *namespace=None*)
Set the L3 settings for the interface using data from the port. ip_cidrs: list of 'X.X.X.X/YY' strings

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)
Plug in the interface.

unplug(*device_name*, *bridge=None*, *namespace=None*, *prefix=None*)
Unplug the interface.

class quantum.agent.linux.interface.**MetaInterfaceDriver**(*conf*)
Bases: quantum.agent.linux.interface.LinuxInterfaceDriver

get_device_name(*port*)

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)

unplug(*device_name*, *bridge=None*, *namespace=None*, *prefix=None*)

class quantum.agent.linux.interface.**NullDriver**(*conf*)
Bases: quantum.agent.linux.interface.LinuxInterfaceDriver

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)

unplug(*device_name*, *bridge=None*, *namespace=None*, *prefix=None*)

class quantum.agent.linux.interface.**OVSInterfaceDriver**(*conf*)
Bases: quantum.agent.linux.interface.LinuxInterfaceDriver

Driver for creating an internal interface on an OVS bridge.

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)
Plug in the interface.

unplug(*device_name*, *bridge=None*, *namespace=None*, *prefix=None*)
Unplug the interface.

POL New **class** quantum.agent.linux.interface.**OVSLibvirtInterfaceDriver**(*conf*)
Bases: quantum.agent.linux.interface.OVSInterfaceDriver

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)
Plug in the interface.

Param network_id: network identifier.

Param port_id: port identifier.

Param device_name: name of the network interface to create.

Param mac_address: MAC address to be set on the new interface.

Param bridge: name of the bridge where the interface will be plug.

Param namespace: namespace name.

Param prefix: namespace prefix.

class quantum.agent.linux.interface.**OVSVethInterfaceDriver**(*conf*)
Bases: quantum.agent.linux.interface.OVSInterfaceDriver

Driver for creating an OVS interface using veth.

DEV_NAME_PREFIX = 'ns-'

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)
Plugin the interface.

unplug(*device_name*, *bridge=None*, *namespace=None*, *prefix=None*)
Unplug the interface.

class quantum.agent.linux.interface.**RyuInterfaceDriver**(*conf*)
Bases: quantum.agent.linux.interface.OVSInterfaceDriver

Driver for creating a Ryu OVS interface.

plug(*network_id*, *port_id*, *device_name*, *mac_address*, *bridge=None*, *namespace=None*, *prefix=None*)
Plug in the interface.

1.3.2 Source: quantum/agent/linux/libvirt_config.py POL New

Configuration for libvirt objects.

Classes to represent the configuration of various libvirt objects and support conversion to/from XML

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetwork**(**kwargs*)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

add_tunnel(*tunnel*)
Set LibvirtConfigNetworkTunnel XML object.

Parameters **tunnel** – LibvirtConfigNetworkTunnel XML object.

format_dom()
Return LibvirtConfigNetwork XML object.

set_bridge(*br*)
Set LibvirtConfigNetworkBridge XML object.

Parameters **br** – LibvirtConfigNetworkBridge XML object.

set_forward(*fwd*)
Set LibvirtConfigNetworkForward XML object.

Parameters **fwd** – LibvirtConfigNetworkForward XML object.

set_ip(*ip*)
Set LibvirtConfigNetworkIP XML object.

Parameters **ip** – LibvirtConfigNetworkIP XML object.

set_portgroup(*portgroup*)
Set LibvirtConfigNetworkPortGroup XML object.

Parameters **portgroup** – LibvirtConfigNetworkPortGroup XML object.

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetworkBridge**(**kwargs*)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

Generate bridge block of XML network configuration.

format_dom()
Return LibvirtConfigNetworkBridge XML object.

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetworkDHCP**(**kwargs*)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

Generate DHCP block of XML network configuration.

format_dom()
Return LibvirtConfigNetworkDHCP XML object.

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetworkForward**(**kwargs*)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

Generate forward block of XML network configuration.

format_dom()
Return LibvirtConfigNetworkForward XML object.

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetworkIP**(**kwargs)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

Generate IP block of XML network configuration.

format_dom()
Return LibvirtConfigNetworkIP XML object.

set_dhcp(*dhcp*)

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetworkPortGroup**(**kwargs)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

Generate port group block of XML network configuration.

add_vport_param(*key*, *value*)
Add a virtual port parameter.

Parameters

- **key** – virtual port key.
- **value** – virtual port value.

format_dom()
Return LibvirtConfigNetworkPortGroup XML object.

set_vlan(*vlan*)
Set VLAN XML object.

Parameters **vlan** – LibvirtConfigNetworkVlan XML object.

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetworkTunnel**(**kwargs)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

format_dom()
Return LibvirtConfigNetworkTunnel XML object.

class quantum.agent.linux.libvirt_config.**LibvirtConfigNetworkVlan**(**kwargs)
Bases: quantum.agent.linux.libvirt_config.LibvirtConfigObject

Generate vlan block of XML network configuration.

add_vlan_tag(*tag*)
Add a VLAN tag.

Parameters **tag** – VLAN tag to add.

format_dom()
Return LibvirtConfigNetworkVlan XML object.

class quantum.agent.linux.libvirt_config.**LibvirtConfigObject**(**kwargs)
Bases: object

Taken from nova/virt/libvirt/config.py.

format_dom()

parse_dom(*xmldoc*)

parse_str(*xmlstr*)

to_xml(*pretty_print=True*)

1.3.3 Source: quantum/agent.linux/libvirt_lib.py POL New

class quantum.agent.linux.libvirt_lib.**Libvirt**(*read_only=False*, *libvirt_type='qemu'*)
Taken from nova/virt/libvirt/config.py.

has_min_version(*ver*)

init_host(*host*)

uri

quantum.agent.linux.libvirt_lib.**patch_tpool_proxy**()
eventlet.tpool.Proxy doesn't work with old-style class in __str__() or __repr__() calls. See bug #962840 for details. We perform a monkey patch to replace those two instance methods.

1.3.4 Source: quantum/agent/rpc.py

class quantum.agent.rpc.**NotificationDispatcher**
Bases: object

run_dispatch(*handler*)

class quantum.agent.rpc.**PluginApi**(*topic*)
Bases: quantum.openstack.common.rpc.proxy.RpcProxy

Agent side of the rpc API.

API version history: 1.0 - Initial version.

BASE_RPC_API_VERSION = '1.0'

get_device_details(*context*, *device*, *agent_id*)

POL New **get_network_binding**(*context*, *network_id*, *agent_id*)
Abstract method to retrieve the binding between a virtual network and its physical realization.

Parameters

- **context** – RPC context.
- **network_id** – network identifier.
- **agent_id** – agent identifier.

Returns requested network binding.

tunnel_sync(*context*, *tunnel_ip*)

update_device_down(*context*, *device*, *agent_id*)

update_device_up(*context*, *device*, *agent_id*)

quantum.agent.rpc.**create_consumers**(*dispatcher*, *prefix*, *topic_details*)
Create agent RPC consumers.

Parameters

- **dispatcher** – The dispatcher to process the incoming messages.
- **prefix** – Common prefix for the plugin/agent message queues.
- **topic_details** – A list of topics. Each topic has a name and a operation.

Returns A common Connection.

1.3.5 Source: quantum/plugins/ovs_quantum_plugin.py

POL Mod **class** quantum.plugins.openvswitch.ovs_quantum_plugin.**AgentNotifierApi**(*topic*)
Bases: quantum.openstack.common.rpc.proxy.RpcProxy

Agent side of the linux bridge rpc API.

API version history: 1.0 - Initial version.

POL modification: send notifications for two new events (network_create, port_create).

BASE_RPC_API_VERSION = '1.0'

POL New **network_create**(*context*, *network_id*, *segmentation_id*)
Send network_create event.

Parameters

- **context** – RPC context.
- **network_id** – network identifier.
- **segmentation_id** – segmentation identifier (e.g. VLAN ID).

network_delete(*context*, *network_id*)

POL New **port_create**(*context*, *port*, *network_type*, *segmentation_id*, *physical_network*)
Send port_create event.

Parameters

- **context** – RPC context.
- **port** – port identifier.
- **network_type** – network type (e.g. VLAN, GRE, FLAT).
- **segmentation_id** – segmentation identifier (e.g. VLAN ID).
- **physical_network** – name of physical network.

port_update(*context*, *port*, *network_type*, *segmentation_id*, *physical_network*)

tunnel_update(*context*, *tunnel_ip*, *tunnel_id*)

class quantum.plugins.openvswitch.ovs_quantum_plugin.**OVSQuantumPluginV2**(*configfile=None*)
Bases: quantum.db.db_base_plugin_v2.QuantumDbPluginV2, quantum.db.l3_db.L3_NAT_db_mixin

Implement the Quantum abstractions using Open vSwitch.

Depending on whether tunneling is enabled, either a GRE tunnel or a new VLAN is created for each network. An agent is relied upon to perform the actual OVS configuration on each host.

The provider extension is also supported. As discussed in https://bugs.launchpad.net/quantum/+bug/1023156, this class could be simplified, and filtering on extended attributes could be handled, by adding support for extended attributes to the QuantumDbPluginV2 base class. When that occurs, this class should be updated to take advantage of it.

POL Mod **create_network**(*context*, *network*)
Create a virtual network.

Parameters

- **context** – RPC context.
- **network** – network object.

POL modification: send network_create event to agents.

POL New **create_port**(*context*, *port*)
Create a new port on a virtual network.

Parameters

- **context** – RPC context.
- **port** – port object.

Returns port object.

delete_network(*context*, *id*)

delete_port(*context*, *id*, *l3_port_check=True*)

get_network(*context*, *id*, *fields=None*)

get_networks(*context*, *filters=None*, *fields=None*)

setup_rpc()

supported_extension_aliases = ['provider', 'router']

update_network(*context*, *id*, *network*)

update_port(*context*, *id*, *port*)

class quantum.plugins.openvswitch.ovs_quantum_plugin.**OVSRpcCallbacks**(*rpc_context*, *notifier*)
Bases: quantum.db.dhcp_rpc_base.DhcpRpcCallbackMixin

RPC_API_VERSION = '1.0'

create_rpc_dispatcher()
Get the rpc dispatcher for this manager.

If a manager would like to set an rpc API version, or support more than one class as the target of rpc messages, override this method.

get_device_details(*rpc_context*, ***kwargs*)
Agent requests device details

POL New **get_network_binding**(*rpc_context*, ***kwargs*)
Retrieve the binding between a virtual network and its physical realization.

Parameters

- **rpc_context** – RPC context.
- **kwargs** – additional arguments.

Returns requested network binding.

tunnel_sync(*rpc_context*, ***kwargs*)
Update new tunnel.

Updates the datbase with the tunnel IP. All listening agents will also be notified about the new tunnel IP.

update_device_down(*rpc_context*, ***kwargs*)
Device no longer exists on agent

1.3.6 Source: quantum/plugins/openvswitch/common/config.py POL Mod

POL modifications:

- added local_interface parameter to ovs_opts.
- added libvirt_type parameter to agent_opts.

1.3.7 Source: quantum/plugins/openvswitch/agent/ovs_libvirt_quantum_agent.py POL New

class quantum.plugins.openvswitch.agent.ovs_libvirt_quantum_agent.**OVSQuantumAgent**(*local_ip*, *local_interface*, *bridge_mapping*, *root_helper*, *polling_interval*, *reconnect_interval*, *rpc*, *enable_tunneling*, *libvirt_type*)

Bases: object

Implements OVS-based tunneling, VLANs and flat networks.

Two local bridges are created: an integration bridge (defaults to 'br-int') and a tunneling bridge (defaults to 'br-tun'). An additional bridge is created for each physical network interface used for VLANs and/or flat networks.

All VM VIFs are plugged into the integration bridge. VM VIFs on a given virtual network share a common "local" VLAN (i.e. not propagated externally). The VLAN id of this local VLAN is mapped to the physical networking details realizing that virtual network.

For virtual networks realized as GRE tunnels, a Logical Switch (LS) identifier and is used to differentiate tenant traffic on inter-HV tunnels. A mesh of tunnels is created to other Hypervisors in the cloud. These tunnels originate and terminate on the tunneling bridge of each hypervisor. Port patching is done to connect local VLANs on the integration bridge to inter-hypervisor tunnels on the tunnel bridge.

For each virtual networks realized as a VLANs or flat network, a veth is used to connect the local VLAN on the integration bridge with the physical network bridge, with flow rules adding, modifying, or stripping VLAN tags as necessary.

MAX_VLAN_TAG = 4094

MIN_VLAN_TAG = 1

RPC_API_VERSION = '1.0'

create_rpc_dispatcher()

Get the rpc dispatcher for this manager.

If a manager would like to set an rpc API version, or support more than one class as the target of rpc messages, override this method.

daemon_loop(*db_connection_url*)

Main loop.

network_create(*context*, ***kwargs*)
Handle network_create event.

Parameters **kwargs** – event arguments

network_delete(*context*, ***kwargs*)
Handle network_delete event

port_create(*context*, ***kwargs*)
Handle port_create event.

Parameters **kwargs** – event arguments

rpc_loop()

setup_backbone_network()
Create the Backbone Network.

Returns virNetwork Libvirt object for Backbone Network.

setup_rpc()
Setup RPC for communication with Quantum server.

setup_tunnel(*tunnel_ip*, *tunnel_id*)
Add a tunnel to the Backbone Network.

Parameters

- **tunnel_ip** – remote endpoint's IP.
- **tunnel_id** – tunnel identifier associated to this host.

setup_tvd_network(*network_id*, *segmentation_id*)
Create a TVD network.

Parameters

- **network_id** – network identifier.
- **segmentation_id** – segmentation identifier (e.g. VLAN ID)

start_tvd_networks()
Start already defined TVD networks.

tunnel_sync()
Execute tunnels synchronization with Quantum server.

Returns resync flag

tunnel_update(*context*, ***kwargs*)
Handle tunnel_update event.

Parameters **kwargs** – event arguments

quantum.plugins.openvswitch.agent.ovs_libvirt_quantum_agent.**create_network_conf**(*network_type=*
net-
work_id=None,
source-
bridge=None,
vlan_id=None)

Create XML configuration of a network.

Parameters

- **network_type** – backbone/tvd.
- **network_id** – network identifier.

- **sourcebridge** – name of the backbone bridge (tvd network only).
- **vlan_id** – VLAN identifier.

Returns LibvirtConfigNetwork object.

quantum.plugins.openvswitch.agent.ovs_libvirt_quantum_agent.**create_tunnel_conf**(*tunnel_ip*, *tunnel_id*)

Create XML configuration of a tunnel.

Parameters

- **tunnel_ip** – remote endpoint's IP.
- **tunnel_id** – tunnel ID associated to this host.

Returns LibvirtConfigNetworkTunnel object.

quantum.plugins.openvswitch.agent.ovs_libvirt_quantum_agent.**create_tvd_portgroup**(*vlan_id*)

Create XML configuration of a port group with type openvswitch.

Parameters **vlan_id** – VLAN identifier.

Returns LibvirtConfigNetworkPortGroup object.

quantum.plugins.openvswitch.agent.ovs_libvirt_quantum_agent.**main**()

1.4 Man Pages

1.4.1 quantum-server

Quantum Server

Author openstack@lists.launchpad.net

Date 2012-04-05

Copyright OpenStack LLC

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

quantum-server [options]

DESCRIPTION

quantum-server provides a webserver that exposes the Quantum API, and passes all webservice calls to the Quantum plugin for processing.

OPTIONS

--version	show program's version number and exit
-h, --help	show this help message and exit
-v, --verbose	Print more verbose output
-d, --debug	Print debugging output
--config-file=PATH	Path to the config file to use, for example, /etc/quantum/quantum.conf. When not specified (the default), we generally look at the first argument specified to be a config file, and if that is also missing, we search standard directories for a config file. (/etc/quantum/, /usr/lib/pythonX/site-packages/quantum/)

Logging Options: The following configuration options are specific to logging functionality for this program.

--log-config=PATH	If this option is specified, the logging configuration file specified is used and overrides any other logging options specified. Please see the Python logging module documentation for details on logging configuration files.
--log-date-format=FORMAT	Format string for %(asctime)s in log records. Default: %Y-%m-%d %H:%M:%S
--use-syslog	Output logs to syslog.
--log-file=PATH	(Optional) Name of log file to output to. If not set, logging will go to stdout.
--log-dir=LOG_DIR	(Optional) The directory to keep log files in (will be prepended to –logfile)

FILES

plugins.ini file contains the plugin information quantum.conf file contains configuration information in the form of python-gflags.

SEE ALSO

- OpenStack Quantum

BUGS

- Quantum is sourced in Launchpad so you can view current bugs at OpenStack Bugs

PYTHON MODULE INDEX

a

p

INDEX

P

Q

R

S

T

U