[The updated model can be found here: https://docs.google.com/a/noironetworks.com/presentation/d/1Nn1HjghAvk2RTPwvltSrnCUJkidWKWY2ckU7OYAVNpo/edit?pli=1#slide=id.g3831233f8_2912]

Group-based Policy Abstractions for Neutron

Collaborators	
Kyle Mestery, Michael Smith, Mike Dvorkin, Mike Cohen, Sumit Naiksatam - Cisco	Anees Shaikh, Mohammad Banikazemi, Ryan Moats - IBM
Rudra Rugge, Harshad Nakil - Juniper	Chris Wright - Redhat
Ronak Shah, Scott Drennan, Dimitri Stiliadis - Nuage Networks	Nils Swart, Derick Winkworth - Plexxi
Uri Elzur - Intel	Stephen Wong - Midokura
Keith Burns - Noiro Networks	Kanzhe Jiang, Rob Sherwood, Big Switch Networks
Prasad Vellanki, Subra Ongole, Hemanth Ravi - One Convergence	

This blueprint consolidates proposals for extending OpenStack Networking with policy and connectivity abstractions that enable significantly more simplified and application-oriented interfaces than with the current Neutron API model. While the current API derives its model from existing legacy network constructs, such as ports, subnets, routers, etc., the proposed model aims for a highly abstracted interface for application developers to express desired connectivity of application components, and high-level policies governing that connectivity. Given the wide variety of technologies available for cloud networking, this model avoids imposing constraints on the underlying implementation that prescribe a particular approach.

This blueprint represents the merger of two similar proposals from Cisco and IBM tackling this group concept. Where there are differences in the various proposals, the different approaches have been described. In spite of these differences, the different approaches can express equivalent application topologies and connectivity policies to a large extent. Most importantly, they share the common goal of an application-centric, policy-oriented view of

OpenStack Networking.

Scope:

This blueprint proposes policy and grouping abstractions for Neutron that will allow for easier consumption of the networking resources by separate organizations and management systems.

Implementation Overview:

The policy framework described in this blueprint extends the current Neutron model with the notion of policies that can be applied to communication between groups of endpoints. As users look beyond basic connectivity, richer network services and network properties are naturally expressed as as policies. Examples include middlebox traversal (service chaining), QoS, path properties, access control, etc. This proposal suggests a model that allows application administrators to express their networking requirements using group and policy abstractions, with the specifics of which policies are supported and how they are implemented, left to the underlying implementation.

The main advantage of the extensions described in this blueprint is that they allow for an interface to Neutron which is more application-centric than the existing Neutron APIs. For example, the current Neutron API is focused on very network-centric constructs: ports, networks, subnets, routers, and security groups. In the context of networking, these make complete sense. But in the context of cloud applications, these are more cumbersome than needed. Application developers think in different terms -- the policy and group abstractions are designed to allow for the flexibility that an application developer may want when programming something like Neutron.

Additionally, the abstractions described here were designed to offer a large amount of flexibility to plugin authors. Plugins may choose to map groups and policies to existing network-centric primitives. However, if the plugin supports a new form of SDN technology, or a higher level of abstraction, they may leverage the additional flexibility in the model to implement network policy in any way they see fit as well.

The goal of these API extensions is that they become the main interface to Neutron for those deploying applications by providing a simpler interface in which to consume Neutron resources.

Terminology

The following terminology is used in this document to describe the key concepts and objects that the Connectivity Group Extension work brings into the Neutron fold.

Entity	Description
Connectivity Group	Collection of endpoints with a common policy.
Policy	Set of Policy Rule objects describing policy. Policies may be applied between groups, or alternatively, applied to a single group using provide / consume relations. (These alternatives are explained further below).
Policy Rule	Specific <classifier, action> pair, part of a policy.

In many IT environments, there may be multiple administrative domains that control the configuration of the infrastructure. This is particularly true in the typical Enterprise where the network infrastructure is often managed by a separate organization from compute and storage. Often the networking infrastructure resources need to be consumed by the organization deploying applications. This consumption of network resources should allow elasticity and ease of consumption while providing the ability to maintain organizational ownership and control if necessary. The current Neutron networking model attaches application VMs to a logical port. That logical port supports a single VM instance and requires the specification of many desired extensions such as security group membership directly on the logical port. In order to allow for easier consumption of the networking resources by the application and compute organizations, a simple abstraction is needed to present a single elastic entity and hide the details of the underlying network detailed configuration. This blueprint is proposing a Connectivity Group that will contain the configuration of the networking resources such as security group membership, network, and all of the extensions. The application team would deploy these Connectivity Groups as simple named entities. It is also useful to allow the ability for the application team to specify relationships between Connectivity Groups. These relationships would also be named entities provided by the networking organization. These named entities are referred to as policies. The relationships allow applications to dynamically configure the networking infrastructure without being

intimately aware of the underlying configuration.

Endpoints are placed into Connectivity Groups for the purpose of policy application. This is done by user configuration. An Endpoint is typically an individual MAC and/or IP address belonging to a vNic/Nic but may also be a group of VMs such as all VMs belonging to a particular subnet. It should be noted that a single VM may be viewed as multiple Endpoints if it has multiple vNICs. Additionally, for the ease of an initial implementation, a Connectivity Group may be defined by the existing Neutron network concept. Defining a group of VMs as an Endpoint is particularly useful for representing an external network like the public Internet or an endpoint representing all other traffic destinations.

Policy definition:

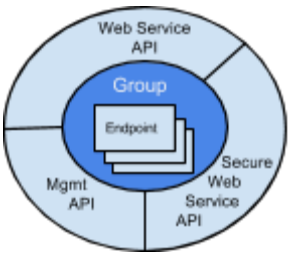
The policy determines connectivity between groups and how the traffic will be treated. It contains a classifier that contains a description of the network traffic to which the policy applies. The classifier is expressed as a list of L4 ports and protocol. The resulting action taken on traffic matching the classifier may be in the form of security actions such as permit / deny, QoS actions, and/or redirection to a service chain. In the future, this classifier may be easily extended to cover additional behaviors as well.

Two different approaches to Policy definition:

The two blueprints that have been consolidated to form this blueprint differed slightly in the way that the policy is expressed and applied between the groups.

Policies applied as a group API

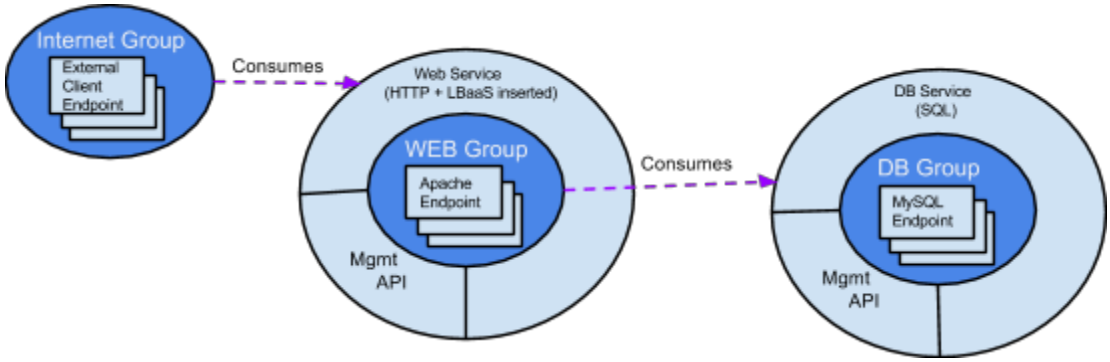
In the first approach, the policy is presented by the group as a service to be consumed by other groups. In other words, the group is providing the service defined by the policy. A given group may provide one or more policies just as it may provide one or more services.



Policy as a Group API

The policy may have constraints defined on what groups may consume the service but for the most part, the provider of the service is unaware of who is consuming the service. This is

very similar to how an application exposes a northbound API. For example, an application server may expose a REST API as the method for its application function to be consumed. The consumers are also unaware of any services that are defined in the policy. For instance, a service may require that a loadbalancer and firewall be inserted. This would be defined within a service chain inside the policy and hidden to the consumer.

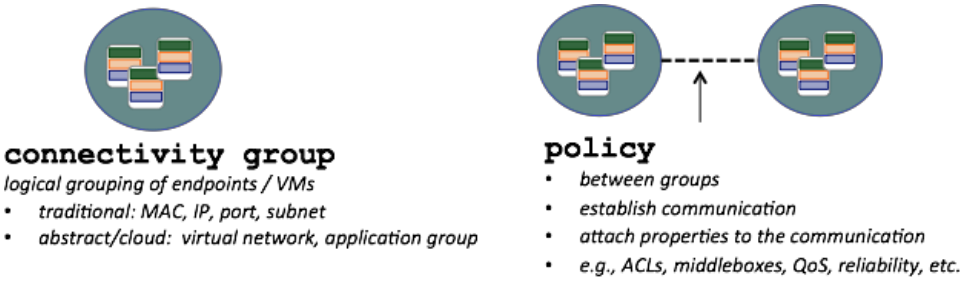


Consuming Group Policies

The consumer of the service is also largely unaware of who exactly is providing a service. For instance, a single policy may be provided by more than 1 group and each group consists of many endpoints. Again, this is very similar to how applications operate today. By modeling the policy as a provided service, this model allows the service to be defined in a single policy that can be shared across many consuming groups. It also allows the application developer to define the service provided by his application independent of the exact method of deployment. This allows others to more easily reuse application components within their multi-tier applications.

Policies applied between groups

In the second approach, the policy is defined between a pair of groups and provides an intuitive way to express allowed communication, as well as properties of that communication. as an explicit connectivity policy between a pair of groups. The policy content is identical to the application-centric policy approach, i.e., including network service insertion, QoS or reliability goals, etc. This view of policy takes is a more explicit way of viewing relationships between groups. Once a policy is defined and named, it can then be applied between multiple sets of groups. In this model, the exact groups that are providing a service and consuming a service are known explicitly whereas in the above approach, the exact groups are implicit based on producer and consumer relationships.

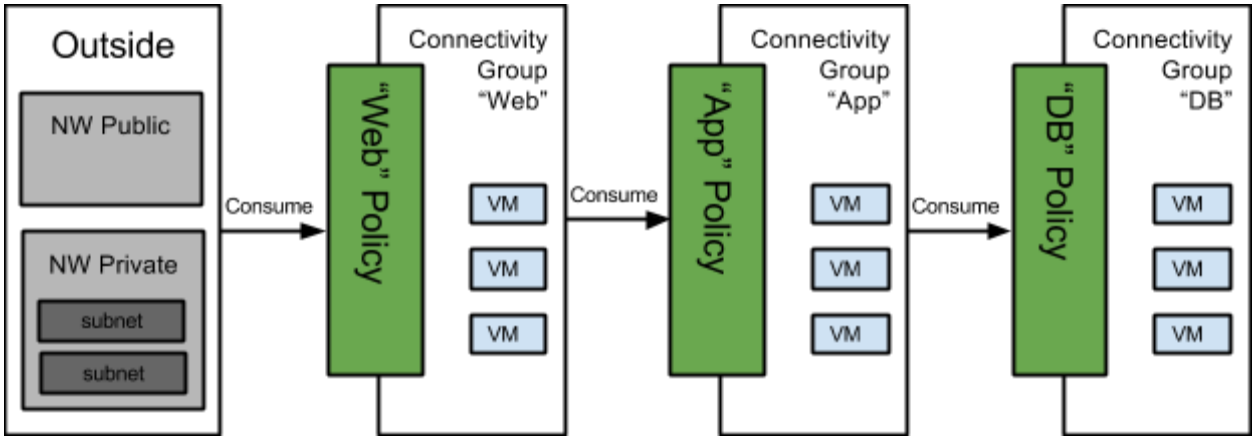


Use Cases:

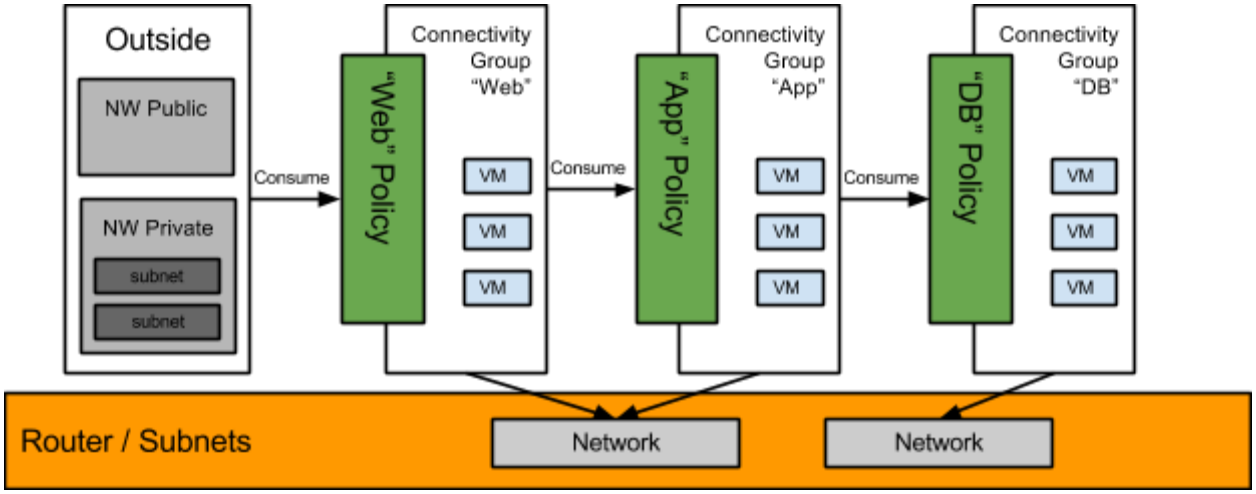
3-tier Application with Security Policies

A specific use case example for such the Connectivity Groups abstraction is a 3-tier Web/App/Database application. In this use case, a tenant creates a Connectivity Group for each tier. Within each Connectivity Group, the network configuration is specified such as the network and security group membership as well as other network configurations. Such a configuration will allow the Web tier to communicate with the App tier but not the Database tier. It does this by specifying security groups and security group rules within the Connectivity Groups.

The following diagrams show how this use case maps into the terminology represented here.

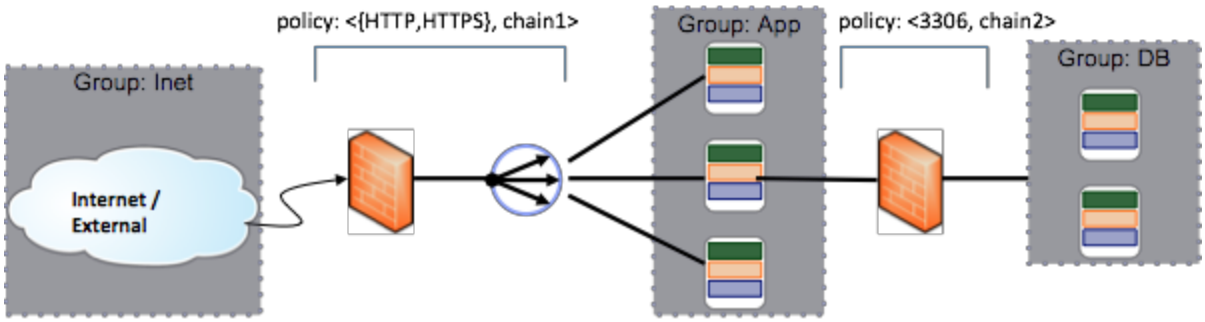


The existing Neutron constructs such as Networks, Routers, and Subnets can still be used directly by the user if so desired. In this case, the new constructs introduced by this blueprint will be rendered onto the existing constructs. The diagram below shows how these new constructs could be mapped into existing Neutron objects. This is used to show a correlation, but is not required by the underlying implementation of the APIs.



Tiered application with service insertion / chaining

Consider another example, but with policies applied between groups to specify that certain traffic types should pass through a defined set of network services. The diagram below illustrates this use case.



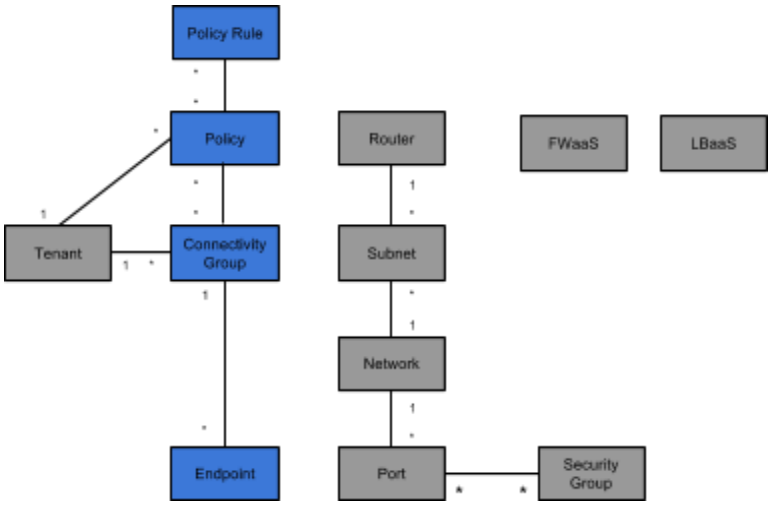
In this example groups have been defined corresponding to external or public Internet endpoints, application cluster, and database cluster. Policies consisting of rules containing <classifier, action> pairs have been defined between groups. In this example, Web traffic from the external network must pass through a firewall and ADC (load balancer or application delivery controller) before reaching the application cluster. Similarly, there is a policy for traffic destined for the database tier which also requires processing by another firewall. In this example the service chains have been abstracted to simple names.

Mapping to existing Neutron objects is straightforward here. These groups could be mapped to virtual networks and subnets, and the individual services in each chain can be configured using existing Neutron service configuration models (e.g., FWaaS and LBaaS). As mentioned

above, however, such a mapping is allowed, but not required, by this model.

Data Model Changes:

We expect some new data model changes to be introduced. The following diagram shows the data model changes required.



Neutron Object Model (grey boxes already exist)

Configuration variables:

The Connectivity Group Extension APIs do not introduce any new configuration variables.

API's:

The Connectivity Group Extension introduces new objects into the Neutron Object Model. These new objects will have appropriate APIs, listed below.

Object	**Verb**	**URI**
Endpoint	GET	/v1.0/endpoints/
Endpoint	GET	/v1.0/endpoints/*endpoint_id*
Endpoint	POST	/v1.0/endpoints
Endpoint	PUT	/v1.0/endpoints/*endpoint_id*
Endpoint	DELETE	/v1.0/endpoints/*endpoint_id*
Connectivity Group	GET	/v1.0/connectivity_groups/
Connectivity Group	GET	/v1.0/connectivity_groups/*connectivity_group*

		_id
Connectivity Group	POST	/v1.0/connectivity_groups
Connectivity Group	PUT	/v1.0/connectivity_groups/*connectivity_group_id*
Connectivity Group	DELETE	/v1.0/connectivity_groups/*connectivity_group_id*
Policy	GET	/v1.0/policies/
Policy	GET	/v1.0/policies/*policy_id*
Policy	POST	/v1.0/policies
Policy	PUT	/v1.0/policies/*policy_id*
Policy	DELETE	/v1.0/policies/*policy_id*
Policy Rule	GET	/v1.0/policy_rules/
Policy Rule	GET	/v1.0/policy_rules/*policy_rule_id*
Policy Rule	POST	/v1.0/policy_rules
Policy Rule	PUT	/v1.0/policy_rules/*policy_id*
Policy Rule	DELETE	/v1.0/policy_rules/*policy_id*

The new API will return error messages for failures occurring during API operations. Standard 4xx HTTP error codes are used as return values to indicate problems with the request sent by the client.

Error	**Description**
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
409	Conflict
413	Over Limit

422	Immutable
500	Internal Server Error
503	Service Unavailable

Some example API calls are shown below:

Example 1. Create Policy Rule: JSON Request

```
POST v1.0/policy_rules.json
Content-Type: application/json
Accept: application/json
{
      "name": "web-rule",
      "direction": "ingress",
      "port_range_min": 80,
      "port_range_max": 80,
      "protocol": "tcp"
      "ethertype": "IPv4"
}
```

Example 2. Create Policy Rule: JSON Response

```
"status": "201"
"content-length": "153"
"content-type": "application/json;
{
      "name": "web-rule",
      "id": "850d3f42-f76a-4f8b-b1cf-5836fc0be940",
      "direction": "ingress",
      "port_range_min": 80,
      "port_range_max": 80,
      "protocol": "tcp"
      "ethertype": "IPv4",
      "policy_ids": [],
      "tenant_id": "f667b69e4d6749749ef3bcba7351d9ce"
```

```
}
```

Example 3. Create Policy: JSON Request

```
POST v1.0/policy.json
Content-Type: application/json
Accept: application/json
{
      "name": "web-policy",
      "policy_rule_ids": ["web-rule"]
}
```

Example 4. Create Policy: JSON Response

```
"status": "201"
"content-length": "153"
"content-type": "application/json;
{
      "name": "web-policy",
      "id": "940d3f42-f76a-4f8b-b1cf-5836fc0be940",
      "policy_rule_ids": ["web-rule"],
      "tenant_id": "f667b69e4d6749749ef3bcba7351d9ce"
}
```

Open Source Reference Implementation

The Group-based Policy Abstraction APIs will be implemented in an Open Source plugin to provide a reference implementation for other plugins to follow, as well as to ensure their adoption. We intend for these APIs to become core Neutron APIs, so ensuring they are a part of an Open Source plugin is the first step towards making this happen. We will implement these in the Modular Layer 2 (ML2) plugin, as in Icehouse the Open vSwitch and Linuxbridge plugins are being deprecated.

Plugin Interface:

To utilize the Connectivity Group Extension APIs, plugins will need to be modified to support these APIs. Plugins that do not support these extension APIs will not require any modification.

Dependencies:

No additional dependencies are required to support the new APIs.

CLI Requirements:

The following CLI commands will be added to the neutron client to support the CGE API extensions:

- Endpoints
    - neutron endpoint-create
    - neutron endpoint-delete
    - neutron endpoint-update
    - neutron endpoint-list
    - neutron endpoint-show
- Connectivity Groups
    - neutron connectivitygroup-create
    - neutron connectivitygroup-delete
    - neutron connectivitygroup-update
    - neutron connectivitygroup-list
    - neutron connectivitygroup-show
- policies
    - neutron policy-create
    - neutron policy-delete
    - neutron policy-update
    - neutron policy-list
    - neutron policy-show
- policy rules
    - neutron policy-rule-create
    - neutron policy-rule-delete
    - neutron policy-rule-update
    - neutron policy-rule-list
    - neutron policy-rule-show

Horizon Requirements:

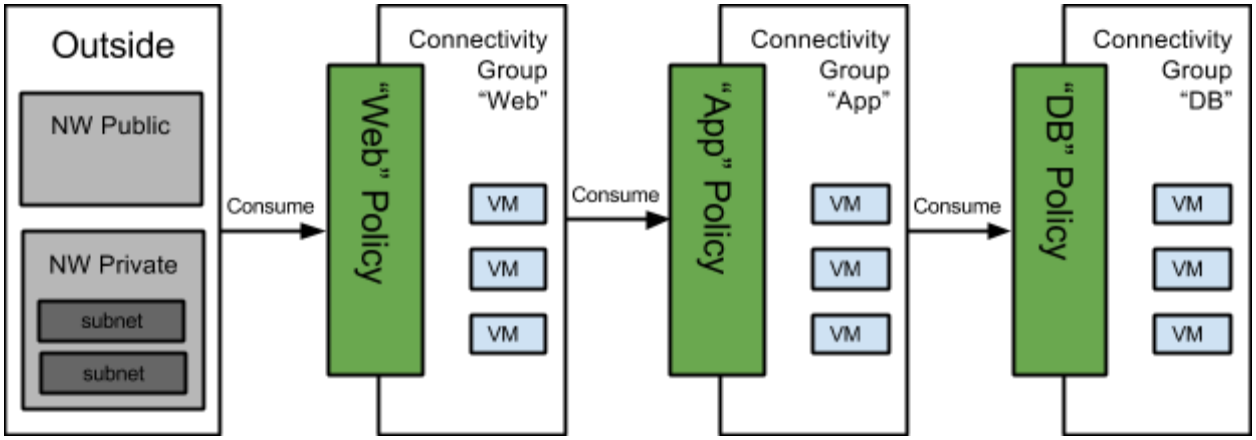
We would like to add support for these new APIs into Horizon. This would allow administrators the option of utilizing Horizon interface to configure Connectivity Groups, Endpoints, and policies.

Usage Examples:

The intent of the Connectivity Group API extensions is to simplify usage of Neutron from an application point of view. Here we present two use cases.

Use Case 1:

An example of using the new APIs is shown below. We will refer to the following picture of a typical 3-tier web application deployment for an example of how this will look. This is shown below.



In the example above, the application will first create policy rules:

neutron policy-rule-create web-rule --direction ingress --protocol tcp --port 80

neutron policy-rule-create all-rule --direction ingress --protocol tcp --port all

neutron policy-rule-create db-rule --direction ingress --protocol tcp --port 3306

Next, the application will create policies:

neutron policy-create web --policy-rule web-rule

neutron policy-create app --policy-rule all-rule

neutron policy-create db --policy-rule db-rule

Next, Connectivity Groups are created, specifying how things are connected:

neutron connectivitygroup-create DB --provide db

neutron connectivitygroup-create APP --provide app --consume db

neutron connectivitygroup-create WEB --provide web --consume app

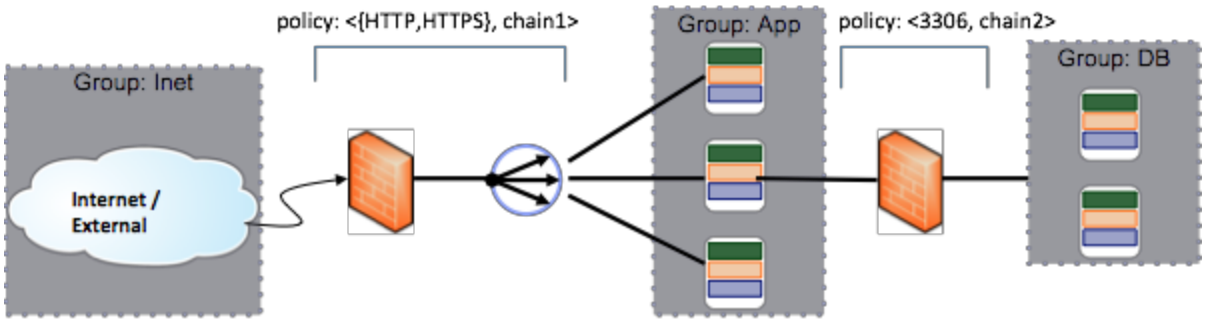
neutron connectivitygroup-create OUTSIDE --consume web

Endpoints will be created implicitly when Nova creates VMs. However, the Connectivity Group API allows for the creation, deletion and update of Endpoints if the application itself desires to perform those operations.

The above will implicitly create the appropriate Neutron constructs and build a compatible network for the 3-tier application to use, utilizing the Connectivity Group Extension constructs. Further, the application will not need to understand or know about network specifics such as ports, networks, or subnets. The Connectivity Group Extension API will allow the application to express it's connectivity desires in more natural terms.

Use Case 2:

In the second usage example, consider the tiered application with service insertion / chaining case discussed earlier:



In the example above, the application will first create the Connectivity Groups and Policy Rules:

neutron connectivitygroup-create Inet --external

neutron connectivitygroup-create App

neutron connectivitygroup-create DB

Policy Rules are defined by specifying the traffic classifier and the action to be performed:

neutron policy-rule-create policyrule-web --protocol tcp --port 80,443 --action chain1

neutron policy-rule-create policyrule-db --protocol tcp --port 3306 --action chain2

Policies are then created by specifying the source and destination Connectivity Groups (policy

endpoints) and the Policy Rules for each policy.

> *neutron policy-create policy-web-ingress --policy-endpoints Inet,App --policy-rule policyrule-web*
>
> *neutron policy-create policy-db-ingress --policy-endpoint App,DB --policy-rule policyrule-db*

Similar to the first use case the Endpoints will be created implicitly when Neutron creates VMs. and the Connectivity Group API allows for the creation, deletion and update of Endpoints if the application itself desires to perform those operations.

The service insertion / service chaining action type can be further defined as a named sequence of identifiers that correspond to virtual appliances and their associated configurations. The details of each action type will be further defined through the course of developing the Group-based policy extension.

The above will implicitly create the appropriate Neutron constructs and build a compatible network for the 3-tier application to use, utilizing the Connectivity Group Extension constructs.

Test Cases:

Tempest tests will be added to handle the new extension APIs proposed here. These will be run with the ML2 plugin and the Open vSwitch MechanismDriver.

More detailed design document starting from next page.

Detailed Design

Here we try to harden the design and pave the way for a prototype implementation.

The design is currently being reworked here:
https://docs.google.com/a/noironetworks.com/presentation/d/1Nn1HjghAvk2RTPwvItSrnCUJkidWKWY2ckU7OYAVNpo/edit#slide=id.g1d6aae2d8_5673

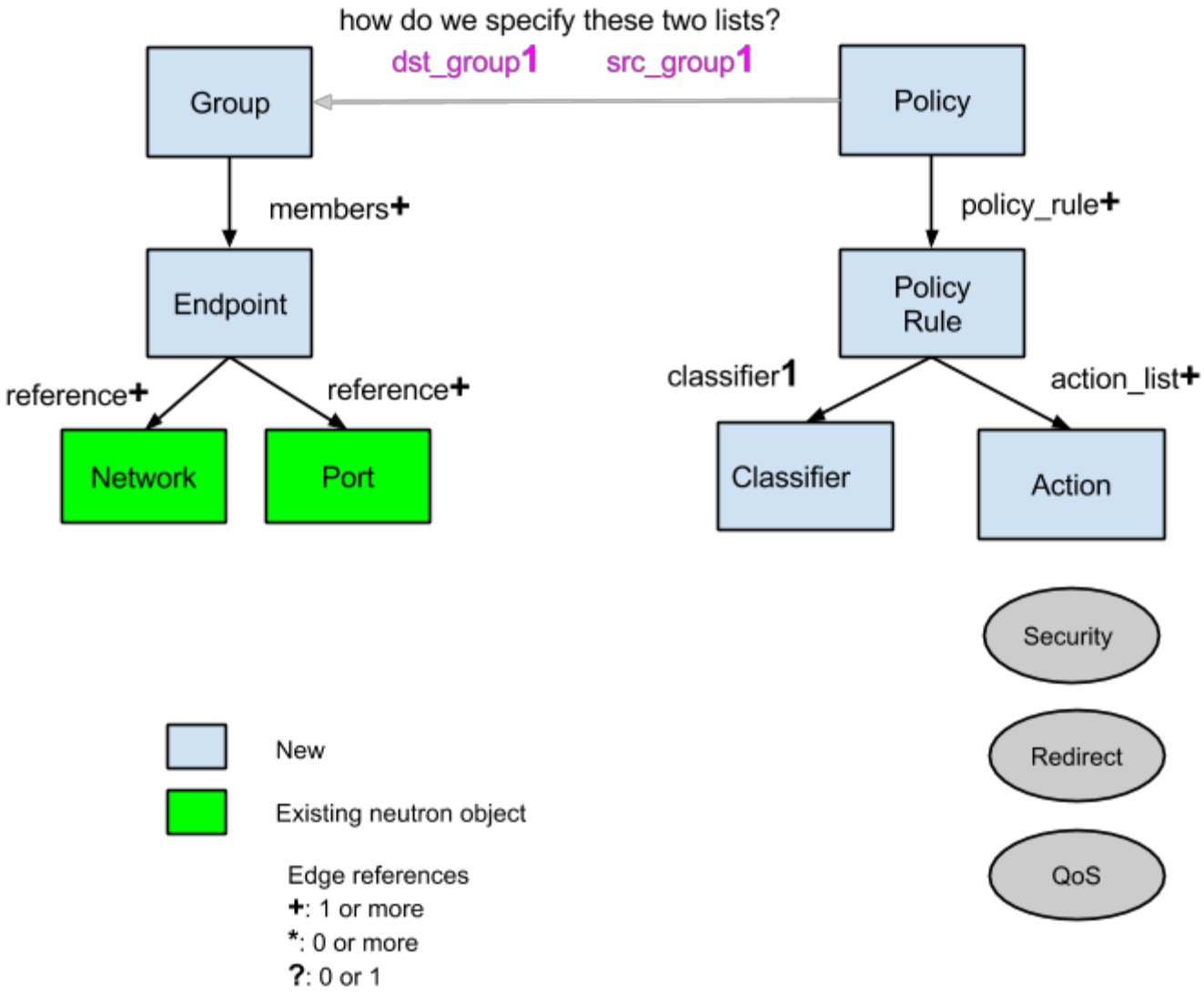
Defining Attributes

New resources:

- group
  - endpoints
- policy_rule
  - classifier
  - list of actions
- policy
  - groups
  - policy_rules

Taxonomy

https://docs.google.com/drawings/d/1HYGUSnxcx_8wkCAwE4Wtv3a30JstOBPyuknf7UnJMp0/edit?usp=sharing



Endpoints and Groups

Endpoints are first class Neutron objects and a group is made of one or more endpoints. It is possible (whether desirable/required or not is to be discussed) that endpoints of a group can be of different types including ports and networks. Note that an endpoint can be a network.

Group Attributes:

Attribute	Type	Required	CRUD	Default	Validation Constraints	Notes
id	uuid-str	N/A	R	generated	N/A	
name	String	No	CRU	None	N/A	
tenant_id	uuid-str	No	CR	from Auth. token	N/A	
members	List (uuid-str)	Yes	CRU	N/A	N/A	list of endpoints

Endpoint Attributes:

Attribute	Type	Required	CRUD	Default	Validation Constraints	Notes
id	uuid-str	N/A	R	generated	N/A	
name	String	No	CRU	None	N/A	
tenant_id	uuid-str	No	CR	from Auth. token	N/A	
type	String	No	CR	“network”	N/A	Currently either “port” or “network”
reference	uuid-str	Yes	CRU	N/A	N/A	Currently required

Policies

Policy Attributes:

Attribute	Type	Required	CRUD	Default	Validation Constraints	Notes
id	uuid-str	N/A	R	generated	N/A	
name	String	No	CRU	None	N/A	

tenant_id	uuid-str	No	CR	from Auth. token	N/A	
src_group	List (uuid-str)	Yes	CR	N/A	N/A	
dst_group	List (uuid-str)	Yes	CR	N/A	N/A	
bidirectional	Boolean	No	CR	False	N/A	
policy_rules	List (uuid-str)	Yes	CRU	N/A	N/A	

Policy Rules Attributes:

Attribute	Type	Required	CRUD	Default	Validation Constraints	Notes
id	uuid-str	N/A	R	generated	N/A	
name	String	No	CRU	None	N/A	
tenant_id	uuid-str	No	CR	from Auth. token	N/A	
classifier	uuid-str	Yes	CR	N/A	N/A	
~~priority (is this needed?)~~	~~integer~~	~~Yes~~	~~CR~~	~~0~~		~~To establish the ordering of policy rules of a policy~~
action_list	List (dict)	Yes	CRU	N/A	N/A	list of dictionary describing {action_type: action}

Classifier Attributes:

Attribute	Type	Required	CRUD	Default	Validation Constraints	Notes

id	uuid-str	N/A	R	generated	N/A	
name	String	No	CRU	None	N/A	
tenant_id	uuid-str	No	CR	from Auth. token	N/A	
type	String	No	CR	N/A	N/A	“unicast” and “broadcast”
ports	List(dict)	No	CR	None	N/A	Sub-ranges of ports [{ "min": "80", “max": "82"}]
protocol	String	No	CR	None	N/A	

Actions:

TODO: a method to query what actions are supported, and functional definitions; methods to update various action values (differs action value types for different action types)

Actions are formatted as a list of dictionary: {action_type: ‘<action-type-string>’, action_value: <action-specific-definition>}

All plugins are required to support the action type ‘security’ as defined below. All other action types are optional.

Action type == “security”

Attribute	Type	Required	Default	Value(s)
action_type	String	Yes	N/A	must be ‘security’
action_value	String	Yes	‘deny’	Currently either “allow” or “drop”

By default, if no ‘security’ action is provided, the default security action of a classifier match is ‘deny’ (packets are dropped)

Conflict Resolution for ‘security’ action type

There are various scenarios where the ‘security’ action values can cause conflict; for example, if a policy contains multiple policy-rules, and a packet matches on multiple classifiers on those rules, each of those matching rules can have ‘security’ action which may contain either ‘allow’ or ‘deny’ values. In case of conflict - that is, matches resulting in both the ‘allow’ and ‘deny’ actions - the ‘**allow**’ action will be taken

Optional action types

The following two action types were mentioned in this document. Support for these action types is optional. Application can query the list of supported action types via **TODO** method

An example of action type == “qos”

Attribute	Type	Required	Default	Value(s)
action_type	String	Yes	N/A	must be ‘qos’
action_value	List (dict)	Yes	N/A	a ‘qos_type’ to value tuple; for example, {‘qos_class’, {‘assured-forwarding’, ‘class-1-med-drop’} and/or ‘{‘rate-limit’, {‘tc-rate’:val, ‘tc-latency’: val, ‘tc-burst’:val }

An example of action type == “redirect”

'redirect' action is used to forward or replicate traffic to other destinations - destination can be another endpoint group, a service chain, a port, or a network. Note that 'redirect' action type can be used with other forwarding related action type such as 'security'; therefore, it is entirely possible that one can specify {'security':'deny'} and still do {'redirect':{'uuid-1', 'uuid-2'...}. And in case of {‘security’:’allow’}, traffic is sent to both the intended endpoint group as well as the

redirect destination. Note that the destination specified on the list CANNOT be the endpoint-group who provides this policy. Also, in case of destination being another endpoint-group, the policy of this new destination endpoint-group will still be applied"

Attribute	Type	Required	Default	Value(s)
action_type	String	Yes	N/A	must be ‘redirect’
action_value	List (uuid-str)	Yes	N/A	can be another endpoint group, or non-group object such as service chain

Heat template in YAML

```
heat_template_version: 2014-01-14

description: >

  YAML template to demonstrate Group Policy resources


# EndPoints

endpoint_1:

      type: OS::Neutron::Endpoint

      properties:

            name: endpoint_1

            type: OS::Neutron::Net

            reference: "f31739b0-7d53-11e3-baa7-0800200c9a66"


endpoint_2:
```

```
        type: OS::Neutron:Endpoint
        properties:
            name: endpoint_2
            type: OS::Neutron::Net
            reference: "f31739b0-7d53-11e3-baa7-0800200c9a66"

# Groups
l_group:
        type: OS:Neutron::ConnectivityGroup
        properties:
            name: l_group
            members: [ { get_resource: endpoint_1 } ]

r_group:
        type: OS::Neutron::ConnectivityGroup
        properties:
            name: r_group
            members: [ { get_resource: endpoint_2 } ]

# Classifier
web_classifier:
        type: OS::Neutron::Classifier
        properties:
            name: web_classifier
            type: unicast
            ports: [ { min: 80, max: 82 } ]
            protocol: tcp
```

```
# Policy Rule
policy_rule:
    type: OS::Neutron::PolicyRule
    properties:
        name: policy_rule
        classifier: { get_resource: web_classifier }
        action_list: [ { "action_type" : "security", "action_value" : "drop" } ]


# Policy
policy_1:
    type: OS::Neutron::Policy
    properties:
        name: policy_1
        src_group: [ { get_resource: l_group } ]
        dst_group: [ { get_resource: r_group } ]
        bidirectional: yes
        policy_rules: [ { get_resource: policy_rule } ]
```