



OPNFV Copper Project

Release draft (10b1da2)

OPNFV

August 19, 2016

1	Introduction	1
1.1	Overall Goals for Configuration Policy	1
1.2	Copper Release 1 Scope	2
1.3	Copper Release 2 Scope	2
2	Definitions	3
3	Abbreviations	5
4	Use Cases	7
4.1	Implemented as of this release	7
4.2	For further analysis and implementation	8
5	Architecture	13
5.1	Architectural Concept	13
5.2	Architectural Aspects	14
6	Requirements	17
6.1	Resource Requirements	17
6.2	Generic Policy Requirements	18
6.3	Requirements Validation Approach	19

INTRODUCTION

Note: This is the working documentation for the Copper project.

The [OPNFV Copper](#) project aims to help ensure that virtualized infrastructure and application deployments comply with goals of the NFV service provider or the VNF designer/user.

This is the second (“Colorado”) release of the Copper project. The documentation provided here focuses on the overall goals of the Copper project, and the specific features supported in the Colorado release.

1.1 Overall Goals for Configuration Policy

As focused on by Copper, configuration policy helps ensure that the NFV service environment meets the requirements of the variety of stakeholders which will provide or use NFV platforms.

These requirements can be expressed as an *intent* of the stakeholder, in specific terms or more abstractly, but at the highest level they express:

- what I want
- what I don’t want

Using road-based transportation as an analogy, some examples of this are shown below.

Table 1.1: Configuration Intent Example

Who I Am	What I Want	What I Don’t Want
user	a van, wheelchair-accessible, electric powered	someone driving off with my van
road provider	keep drivers moving at an optimum safe speed	four-way stops
public safety	shoulder warning strips, center media barriers	speeding, tractors on the freeway

According to their role, service providers may apply more specific configuration requirements than users, since service providers are more likely to be managing specific types of infrastructure capabilities.

Developers and users may also express their requirements more specifically, based upon the type of application or how the user intends to use it.

For users, a high-level intent can be also translated into a more or less specific configuration capability by the service provider, taking into consideration aspects such as the type of application or its constraints.

Examples of such translation are:

Table 1.2: Intent Translation into Configuration Capability

Intent	Configuration Capability
network security	firewall, DPI, private subnets
compute/storage security	vulnerability monitoring, resource access controls
high availability	clustering, auto-scaling, anti-affinity, live migration
disaster recovery	geo-diverse anti-affinity
high compute/storage performance	clustering, affinity
high network performance	data plane acceleration
resource reclamation	low-usage monitoring

Although such intent to capability translation is conceptually useful, it is unclear how it can address the variety of aspects that may affect the choice of an applicable configuration capability.

For that reason, the Copper project will initially focus on more specific configuration requirements as fulfilled by specific configuration capabilities, and how those requirements and capabilities are expressed in VNF and service design and packaging, or as generic policies for the NFVI.

1.2 Copper Release 1 Scope

OPNFV Brahmputra was the initial OPNFV release for Copper, and achieved the goals:

- Add the OpenStack Congress service to OPNFV, through at least one installer project, through post-install configuration.
- Provide basis tests scripts and tools to exercise the Congress service

1.3 Copper Release 2 Scope

OPNFV Colorado includes the additional features:

- Congress support in the the OPNFV CI/CD pipeline for the JOID and Apex installers, through the following projects being upstreamed to OpenStack: * For JOID, a JuJu Charm for Congress * For Apex, a Puppet Module for Congress
- Congress use case tests integrated into Functest and as manual tests
- Further enhancements of Congress test tools

DEFINITIONS

Table 2.1: Definitions

Term	Meaning
State	Information that can be used to convey or imply the state of something,

e.g. an application, resource, entity, etc. This can include data held inside OPNFV components, “events” that have occurred (e.g. “policy violation”), etc.

- – Event
 - An item of significance to the policy engine, for which the engine has become aware through some method of discovery e.g. polling or notification.

ABBREVIATIONS

Table 3.1: Abbreviations

Term	Meaning
CRUD	Create, Read, Update, Delete (database operation types)
FCAPS	Fault, Configuration, Accounting, Performance, Security
NF	Network Function
SFC	Service Function Chaining
VNF	Virtual Network Function

4.1 Implemented as of this release

4.1.1 DMZ Deployment

As a service provider, I need to ensure that applications which have not been designed for exposure in a DMZ zone, are not attached to DMZ networks.

An example implementation is shown in the Congress use case test “DMZ Placement” (dmz.sh) in the Copper repo under the tests folder. This test:

- Identifies VMs connected to a DMZ (currently identified through a specifically-named security group)
- Identifies VMs connected to a DMZ, which are by policy not allowed to be (currently implemented through an image tag intended to identify images that are “authorized” i.e. tested and secure, to be DMZ-connected)
- Reactively enforces the dmz placement rule by pausing VMs found to be in violation of the policy.

As implemented through OpenStack Congress:

```
dmz_server(x) :-
nova:servers(id=x, status='ACTIVE'),
neutronv2:ports(id, device_id, status='ACTIVE'),
neutronv2:security_group_port_bindings(id, sg),
neutronv2:security_groups(sg, name='dmz') "

dmz_placement_error(id) :-
nova:servers(id, name, hostId, status, tenant_id, user_id, image, flavor, az, hh),
not glancev2:tags(image, 'dmz'),
dmz_server(id) "

execute[nova:servers.pause(id)] :-
dmz_placement_error(id),
nova:servers(id, status='ACTIVE') "
```

4.1.2 Configuration Auditing

As a service provider or tenant, I need to periodically verify that resource configuration requirements have not been violated, as a backup means to proactive or reactive policy enforcement.

An example implementation is shown in the Congress use case test “SMTP Ingress” (smtp_ingress.sh) in the Copper repo under the tests folder. This test:

- Detects that a VM is associated with a security group that allows SMTP ingress (TCP port 25)

- Adds a policy table row entry for the VM, which can be later investigated for appropriate use of the security group, etc

As implemented through OpenStack Congress:

```
smtp_ingress(x) :-
nova:servers(id=x,status='ACTIVE'),
neutronv2:ports(port_id, status='ACTIVE'),
neutronv2:security_groups(sg, tenant_id, sgn, sgd),
neutronv2:security_group_port_bindings(port_id, sg),
neutronv2:security_group_rules(sg, rule_id, tenant_id, remote_group_id,
'ingress', ethertype, 'tcp', port_range_min, port_range_max, remote_ip),
lt(port_range_min, 26),
gt(port_range_max, 24)
```

4.1.3 Reserved Resources

As an NFVI provider, I need to ensure that my admins do not inadvertently enable VMs to connect to reserved subnets.

An example implementation is shown in the Congress use case test “Reserved Subnet” (reserved_subnet.sh) in the Copper repo under the tests folder. This test:

- Detects that a subnet has been created in a reserved range
- Reactively deletes the subnet

As implemented through OpenStack Congress:

```
reserved_subnet_error(x) :-
neutronv2:subnets(id=x, cidr='10.7.1.0/24')

execute[neutronv2:delete_subnet(x)] :-
reserved_subnet_error(x)
```

4.2 For further analysis and implementation

4.2.1 Affinity

Ensures that the VM instance is launched “with affinity to” specific resources, e.g. within a compute or storage cluster. Examples include: “Same Host Filter”, i.e. place on the same compute node as a given set of instances, e.g. as defined in a scheduler hint list.

As implemented by OpenStack Heat using server groups:

Note: untested example...

```
resources:
  servgrp1:
    type: OS::Nova::ServerGroup
    properties:
      policies:
        - affinity
      serv1:
        type: OS::Nova::Server
        properties:
          image: { get_param: image }
          flavor: { get_param: flavor }
```

```

networks:
  - network: {get_param: network}
serv2:
type: OS::Nova::Server
properties:
  image: { get_param: image }
  flavor: { get_param: flavor }
  networks:
    - network: {get_param: network}

```

4.2.2 Anti-Affinity

Ensures that the VM instance is launched “with anti-affinity to” specific resources, e.g. outside a compute or storage cluster, or geographic location. Examples include: “Different Host Filter”, i.e. ensures that the VM instance is launched on a different compute node from a given set of instances, as defined in a scheduler hint list.

As implemented by OpenStack Heat using scheduler hints:

Note: untested example...

```

heat template version: 2013-05-23
parameters:
  image:
    type: string
    default: TestVM
  flavor:
    type: string
    default: ml.micro
  network:
    type: string
    default: cirros_net2
resources:
  serv1:
    type: OS::Nova::Server
    properties:
      image: { get_param: image }
      flavor: { get_param: flavor }
      networks:
        - network: {get_param: network}
      scheduler_hints: {different_host: {get_resource: serv2}}
  serv2:
    type: OS::Nova::Server
    properties:
      image: { get_param: image }
      flavor: { get_param: flavor }
      networks:
        - network: {get_param: network}
      scheduler_hints: {different_host: {get_resource: serv1}}

```

4.2.3 Network Access Control

Networks connected to VMs must be public, or owned by someone in the VM owner’s group.

This use case captures the intent of the following sub-use-cases:

- **Link Mirroring:** As a troubleshooter, I need to mirror traffic from physical or virtual network ports so that I can investigate trouble reports.

- **Link Mirroring:** As a NFVaaS tenant, I need to be able to mirror traffic on my virtual network ports so that I can investigate trouble reports.
- **Unauthorized Link Mirroring Prevention:** As a NFVaaS tenant, I need to be able to prevent other tenants from mirroring traffic on my virtual network ports so that I can protect the privacy of my service users.
- **Link Mirroring Delegation:** As a NFVaaS tenant, I need to be able to allow my NFVaaS SP customer support to mirror traffic on my virtual network ports so that they can assist in investigating trouble reports.

As implemented through OpenStack Congress:

Note: untested example...

```
error :-
nova:vm(vm),
neutron:network(network),
nova:network(vm, network),
neutron:private(network),
nova:owner(vm, vm-own),
neutron:owner(network, net-own),
-same-group(vm-own, net-own)

same-group(user1, user2) :-
ldap:group(user1, g),
ldap:group(user2, g)
```

4.2.4 Storage Access Control

Storage resources connected to VMs must be owned by someone in the VM owner's group.

As implemented through OpenStack Congress:

Note: untested example...

```
error :-
nova:vm(vm),
cinder:volumes(volume),
nova:volume(vm, volume),
nova:owner(vm, vm-own),
neutron:owner(volume, vol-own),
-same-group(vm-own, vol-own)

same-group(user1, user2) :-
ldap:group(user1, g),
ldap:group(user2, g)
```

4.2.5 Resource Reclamation

As a service provider or tenant, I need to be informed of VMs that are under-utilized so that I can reclaim the VI resources. (example from [RuleYourCloud blog](#))

As implemented through OpenStack Congress:

Note: untested example...

```
reclaim_server(vm) :-
ceilometer:stats("cpu_util",vm, avg_cpu),
lessthan(avg_cpu, 1)
```

```
error(user_id, email, vm_name) :-
reclaim_server(vm),
nova:servers(vm, vm_name, user_id),
keystone:users(user_id, email)
```

4.2.6 Resource Use Limits

As a tenant or service provider, I need to be automatically terminate an instance that has run for a pre-agreed maximum duration.

As implemented through OpenStack Congress:

Note: untested example...

```
terminate_server(vm) :-
ceilometer:statistics("duration",vm, avg_cpu),
lessthan(avg_cpu, 1)

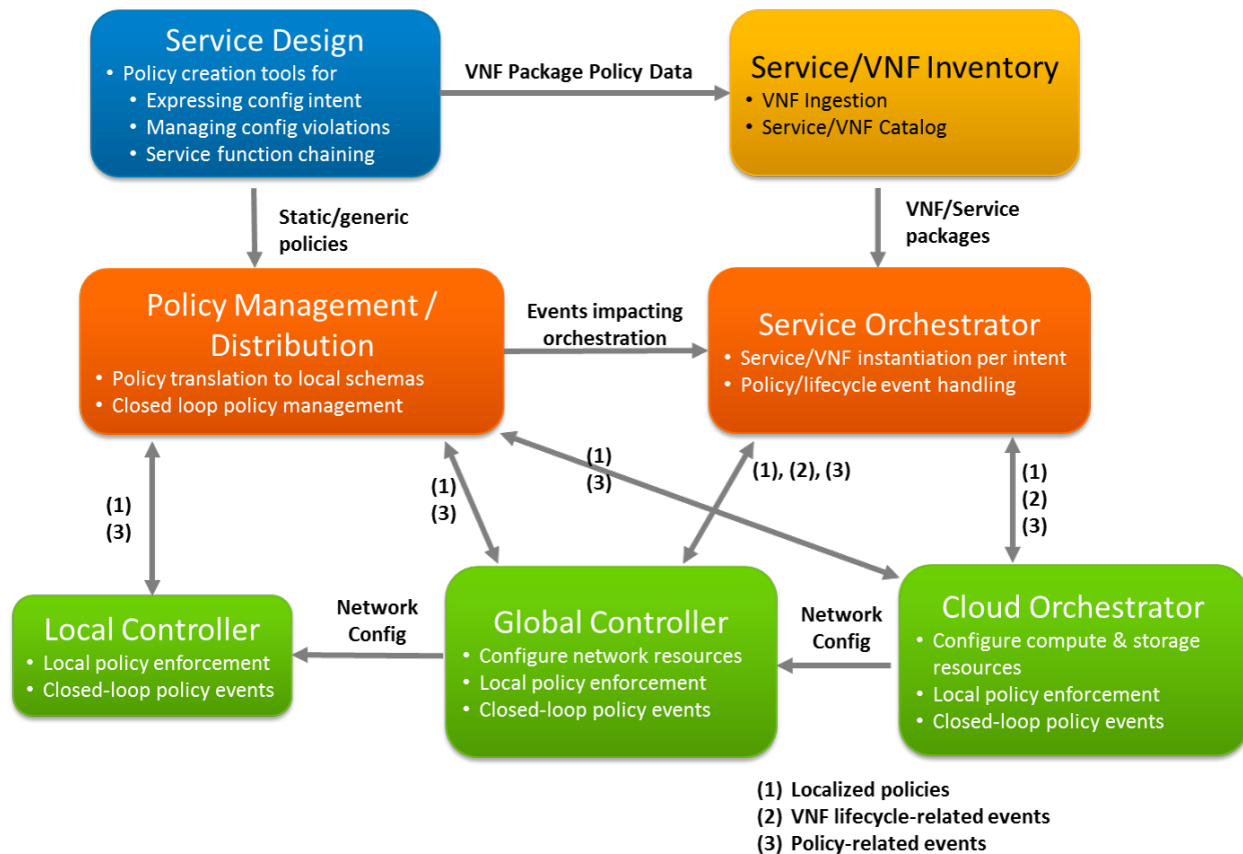
error(user_id, email, vm_name) :-
reclaim_server(vm),
nova:servers(vm, vm_name, user_id),
keystone:users(user_id, email)
```


ARCHITECTURE

5.1 Architectural Concept

The following example diagram illustrates a “relationship diagram” type view of an NFVI platform, in which the roles of components focused on policy management, services, and infrastructure are shown.

This view illustrates that a large-scale deployment of NFVI may leverage multiple components of the same “type” (e.g. SDN Controller), which fulfill specific purposes for which they are optimized. For example, a global SDN controller and cloud orchestrator can act as directed by a service orchestrator in the provisioning of VNFs per intent, while various components at a local and global level handle policy-related events directly and/or feed them back through a closed-loop policy design that responds as needed, directly or through the service orchestrator.



(source of the diagram above: https://git.opnfv.org/cgit/copper/plain/design_docs/images/policy_architecture.pptx)

5.2 Architectural Aspects

- Policies are reflected in two high-level goals
 - Ensure resource requirements of VNFs and services are applied per VNF designer, service, and tenant intent
 - Ensure that generic policies are not violated, e.g. *networks connected to VMs must either be public or owned by the VM owner*
- Policies are distributed through two main means
 - As part of VNF packages, customized if needed by Service Design tools, expressing intent of the VNF designer and service provider, and possibly customized or supplemented by service orchestrators per the intent of specific tenants
 - As generic policies provisioned into VIMs (SDN controllers and cloud orchestrators), expressing intent of the service provider re what states/events need to be policy-governed independently of specific VNFs
- Policies are applied locally and in closed-loop systems per the capabilities of the local policy enforcer and the impact of the related state/event conditions
 - VIMs should be able to execute most policies locally
 - VIMs may need to pass policy-related state/events to a closed-loop system, where those events are relevant to other components in the architecture (e.g. service orchestrator), or some additional data/arbitration is needed to resolve the state/event condition
- Policies are localized as they are distributed/delegated
 - High-level policies (e.g. expressing “intent”) can be translated into VNF package elements or generic policies, perhaps using distinct syntaxes
 - Delegated policy syntaxes are likely VIM-specific, e.g. Datalog (Congress)
- Closed-loop policy and VNF-lifecycle event handling are //somewhat// distinct
 - Closed-loop policy is mostly about resolving conditions that can’t be handled locally, but as above in some cases the conditions may be of relevance and either delivered directly or forwarded to service orchestrators
 - VNF-lifecycle events that can’t be handled by the VIM locally are delivered directly to the service orchestrator
- Some events/analytics need to be collected into a more “open-loop” system which can enable other actions, e.g.
 - audits and manual interventions
 - machine-learning focused optimizations of policies (largely a future objective)

Issues to be investigated as part of establishing an overall cohesive/adaptive policy architecture:

- For the various components which may fulfill a specific purpose, what capabilities (e.g. APIs) do they have/need to
 - handle events locally
 - enable closed-loop policy handling components to subscribe/optimize policy-related events that are of interest
- For global controllers and cloud orchestrators
 - How do they support correlation of events impacting resources in different scopes (network and cloud)
 - What event/response flows apply to various policy use cases
- What specific policy use cases can/should fall into each overall class

- locally handled by NFVI components
- handled by a closed-loop policy system, either VNF/service-specific or VNF-independent

REQUIREMENTS

This section outlines general requirements for configuration policies, per the two main aspects in the Copper project scope:

- Ensuring resource requirements of VNFs and services are applied per VNF designer, service, and tenant intent
- Ensuring that generic policies are not violated, e.g. *networks connected to VMs must either be public or owned by the VM owner*

6.1 Resource Requirements

Resource requirements describe the characteristics of virtual resources (compute, storage, network) that are needed for VNFs and services, and how those resources should be managed over the lifecycle of a VNF/service. Upstream projects already include multiple ways in which resource requirements can be expressed and fulfilled, e.g.:

- OpenStack Nova
 - the **image** feature, enabling “VM templates” to be defined for NFs, and referenced by name as a specific NF version to be used
 - the **flavor** feature, addressing basic compute and storage requirements, with extensibility for custom attributes
- OpenStack Heat
 - the **Heat Orchestration Template** feature, enabling a variety of VM aspects to be defined and managed by Heat throughout the VM lifecycle, notably
 - * alarm handling (requires **Ceilometer**)
 - * attached volumes (requires **Cinder**)
 - * domain name assignment (requires **Designate**)
 - * images (requires **Glance**)
 - * autoscaling
 - * software configuration associated with VM lifecycle hooks (CREATE, UPDATE, SUSPEND, RESUME, DELETE)
 - * wait conditions and signaling for sequencing orchestration steps
 - * orchestration service user management (requires **Keystone**)
 - * shared storage (requires **Manila**)
 - * load balancing (requires Neutron **LBaaS**)

- * firewalls (requires Neutron [FWaaS](#))
- * various Neutron-based network and security configuration items
- * Nova flavors
- * Nova server attributes including access control
- * Nova server group affinity and anti-affinity
- * “Data-intensive application clustering” (requires [Sahara](#))
- * DBaaS (requires [Trove](#))
- * “multi-tenant cloud messaging and notification service” (requires [Zaqar](#))
- OpenStack [Group-Based Policy](#)
 - API-based grouping of endpoints with associated contractual expectations for data flow processing and service chaining
- OpenStack [Tacker](#)
 - “a fully functional ETSI MANO based general purpose NFV Orchestrator and VNF Manager for OpenStack”
- OpenDaylight [Group-Based Policy](#)
 - model-based grouping of endpoints with associated contractual expectations for data flow processing
- OpenDaylight [Service Function Chaining \(SFC\)](#)
 - model-based management of “service chains” and the infrastructure that enables them
- Additional projects that are commonly used for configuration management, implemented as client-server frameworks using model-based, declarative, or scripted configuration management data.
 - [Puppet](#)
 - [Chef](#)
 - [Ansible](#)
 - [Salt](#)

6.2 Generic Policy Requirements

Generic policy requirements address conditions related to resource state and events which need to be monitored for, and optionally responded to or prevented. These conditions are typically expected to be VNF/service-independent, as VNF/service-dependent condition handling (e.g. scale in/out) are considered to be addressed by VNFM/NFVO/VIM functions as described under Resource Requirements or as FCAPS related functions. However the general capabilities below can be applied to VNF/service-specific policy handling as well, or in particular to invocation of VNF/service-specific management/orchestration actions. The high-level required capabilities include:

- Polled monitoring: Exposure of state via request-response APIs.
- Notifications: Exposure of state via pub-sub APIs.
- Realtime/near-realtime notifications: Notifications that occur in actual or near realtime.
- Delegated policy: CRUD operations on policies that are distributed to specific components for local handling, including one/more of monitoring, violation reporting, and enforcement.
- Violation reporting: Reporting of conditions that represent a policy violation.

- Reactive enforcement: Enforcement actions taken in response to policy violation events.
- Proactive enforcement: Enforcement actions taken in advance of policy violation events, e.g. blocking actions that could result in a policy violation.
- Compliance auditing: Periodic auditing of state against policies.

Upstream projects already include multiple ways in which configuration conditions can be monitored and responded to:

- OpenStack [Congress](#) provides a table-based mechanism for state monitoring and proactive/reactive policy enforcement, including data obtained from internal databases of OpenStack core and optional services. The Congress design approach is also extensible to other VIMs (e.g. SDNCs) through development of data source drivers for the new monitored state information. See [Stackforge Congress Data Source Translators](#), [congress.readthedocs.org](#), and the [Congress specs](#) for more info.
- OpenStack [Ceilometer](#) provides means to trigger alarms upon a wide variety of conditions derived from its monitored OpenStack analytics.
- [Nagios](#) “offers complete monitoring and alerting for servers, switches, applications, and services”.

6.3 Requirements Validation Approach

The Copper project will assess the completeness of the upstream project solutions for requirements in scope through a process of:

- developing configuration policy use cases to focus solution assessment tests
- integrating the projects into the OPNFV platform for testing
- executing functional and performance tests for the solutions
- assessing overall requirements coverage and gaps in the most complete upstream solutions

Depending upon the priority of discovered gaps, new requirements will be submitted to upstream projects for the next available release cycle.