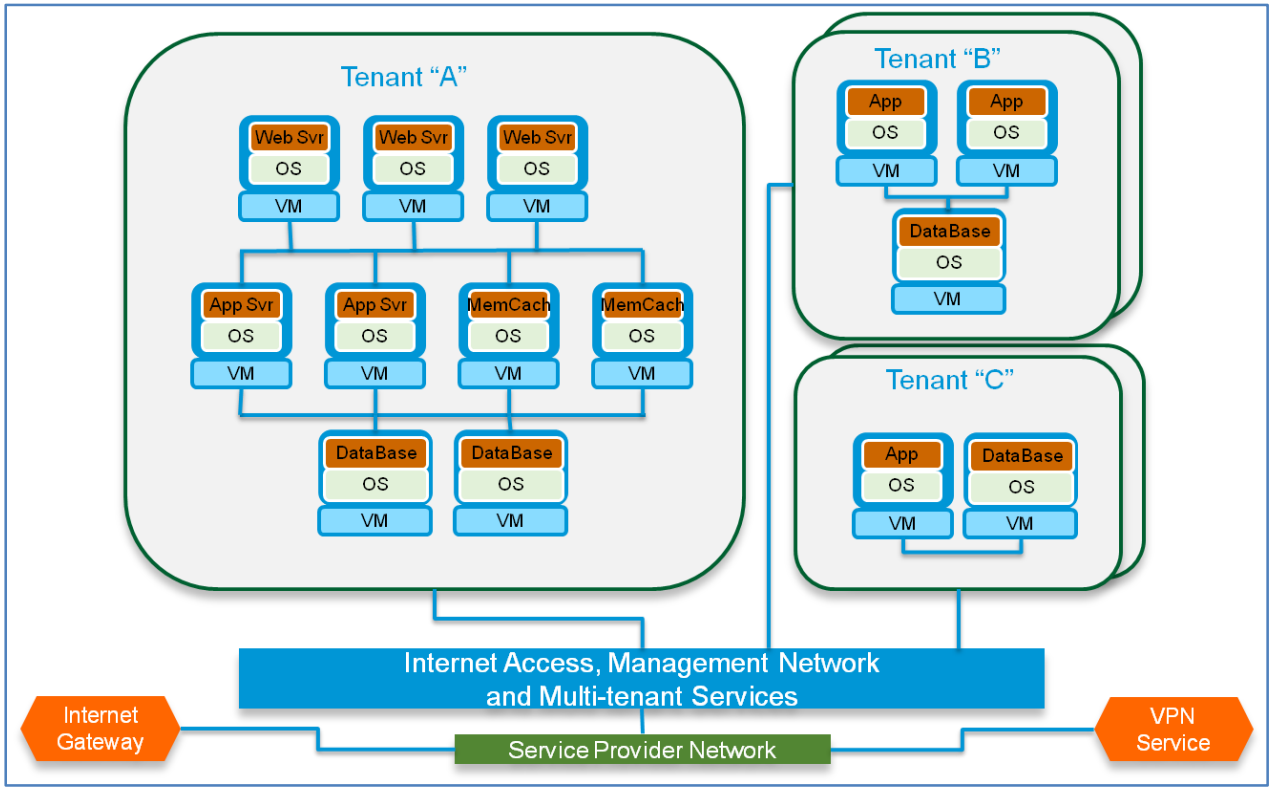
L3 Tenant-facing Abstractions/Model

Motivation

- Routing within the tenant (support multi-tier topologies)
- Overlapping IP addresses
- Support gateways – Internet, VPN
- Support other L3 services – LB, Firewall, Caching, etc.
- Hybrid Cloud (Public + Private)

Further evolve Quantum to be a multi-tenant network service for creating virtual data centers (application specific topologies + network services)



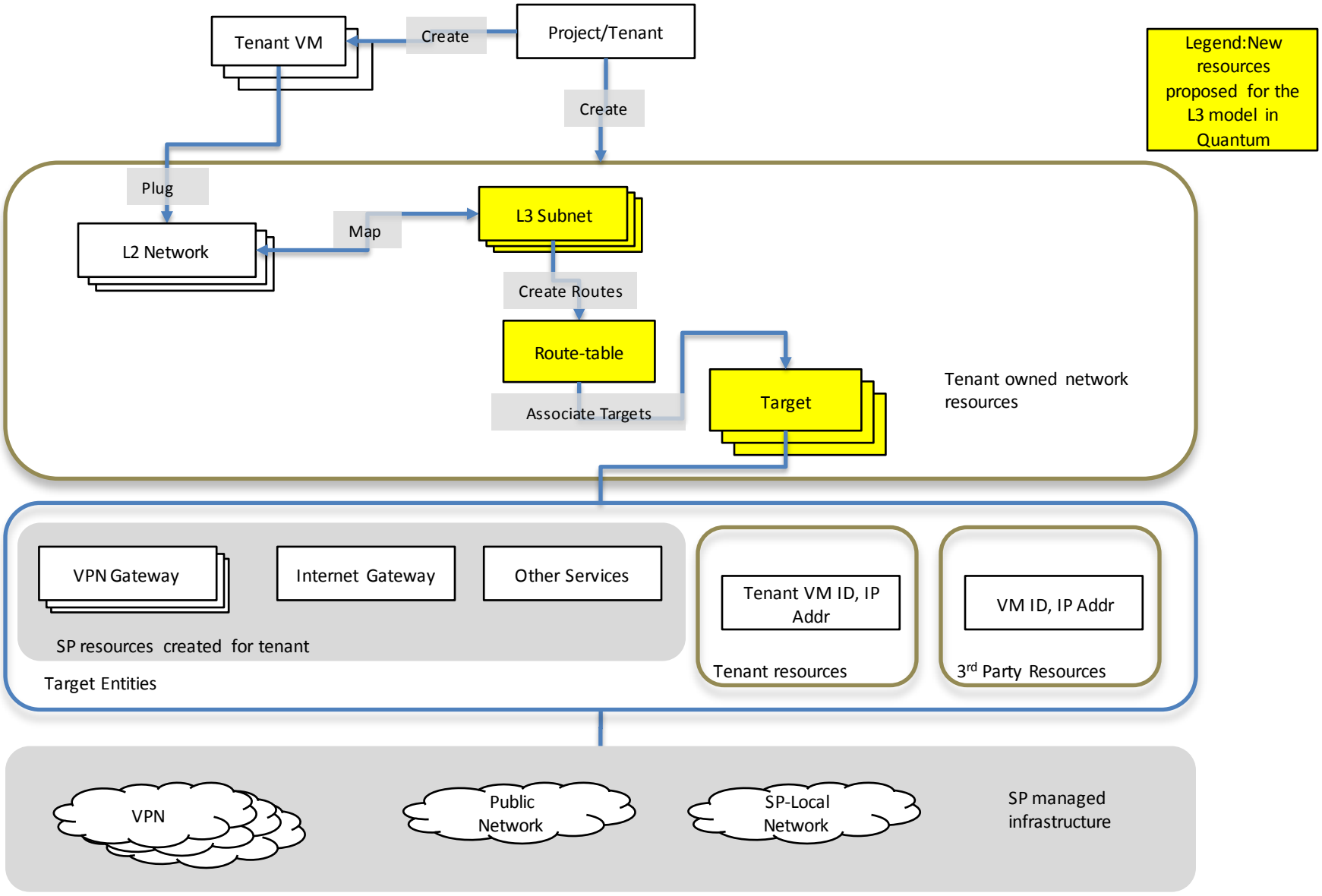
Proposal: Tenant Facing L3 Abstractions

- Tenant
  - An individual customer or a distributed application that requires a set of resources assigned to meet a particular set of functions.
  - Expects networking services from Quantum to isolate and/or connect subsets of the resources
- L3 Abstractions
  - Allow the tenant to request connectivity between its resources at the granularity of subnets and endpoints on a subnet
  - Allow the tenant to request gateway services
  - Allow the tenant to insert it's own gateway services
- Design Goal – Make life easier for the user (user will probably *not* be a network admin!)

The role of a Plugin in Quantum

- Before proceeding further, it's important to note that Quantum adopts a plugin-based approach to realize the abstractions it exposes
- A plugin is a vendor/technology-specific entity which maps the abstractions to the underlying infrastructure
- As such, the Service Provider specific model required to realize the tenant-facing abstractions can be wrapped inside the plugin
- A SP model to represent the SP infra is also desirable, however, as a first step we are proposing only a tenant facing model

Quantum L3 (Tenant) Model Schematic

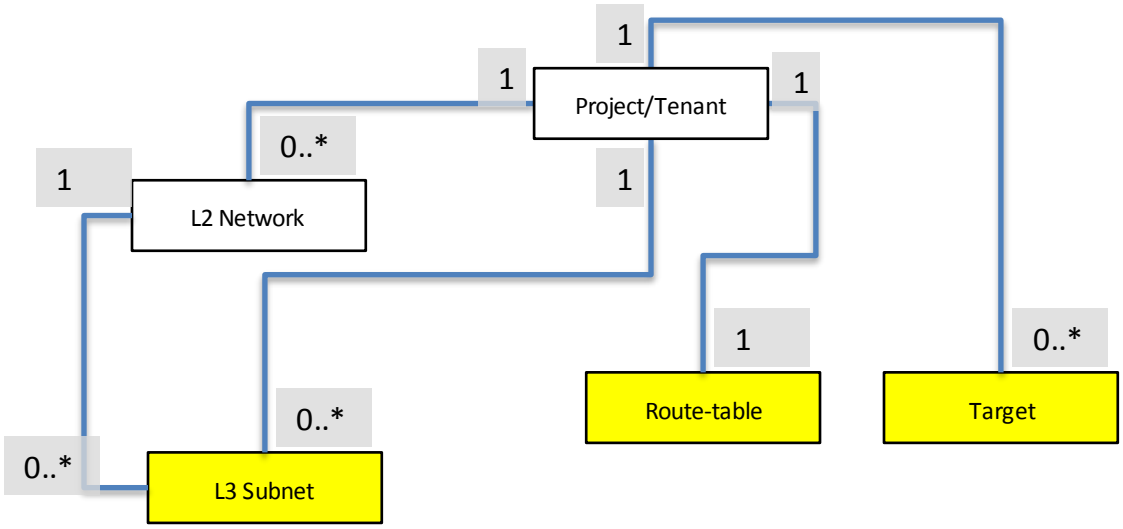


Note: Route-table and Target are *not* the same as routing tables in a router, or a route-target in the BGP context.

Definitions (as used in the current context)

- Subnets
  - A logical subdivision of an IP network expressed using a CIDR notation. Entities residing on a subnet appear to the tenant as directly connected and do not require setting up routes for connectivity (with the ability to override this default behavior).
- Route-tables
  - A Route-table is a tenant instantiated resource that provides operations to manipulate routing entries. One can conceptually think of a Route-table mapping to a logical routing entity.
  - Each row in the table has a Target column, which effectively resolves to a next hop – hence the notion of routes and route-table. (Note: as explained later these route-tables are different from routing tables in routers or Linux hosts.)
- Targets
  - These are service gateways such as an Internet Gateway, VPN, NAT router, a VM which is serving this function, or any other service.

Resources' Relationship



Route-table

A Route-table is a tenant instantiated resource that provides operations to manipulate routing entries. One can conceptually think of a Route-table mapping to a logical routing entity.

Each entry in the Route-table is of the following format:

Source	Destination	Target

Source: Subnet resource ID or CIDR

Destination: Subnet resource ID or CIDR (0.0.0.0 for default)

Target: A Target resource ID such that the SP should be able to resolve the ID to a single endpoint (e.g. IP address)

One Route-table per tenant.

Targets

These are service gateways such as an Internet Gateway, VPN, NAT router, a VM which is serving this function, or any other service.

The target can take one of the following values:

1. A reference (i.e. UUID) to an explicitly created target resource. This target resource could point to one of the following:
   a. A gateway service (Internet , VPN, other) provided by the SP.
   b. A resource belonging to the same tenant or another tenant providing some service (e.g. a NAT VM).

   (Details on how to create the target resource are in the following slides.)

2. A reference (i.e. UUID) to the subnet in the Destination column of the route-table (this is to account for the direct connection between two subnets, or to IP addresses on the same subnet).

Creating a Target to point to a SP managed gateway

The tenant queries the SP for the available target types by calling:

```
list_target_types(tenant_id)
```

The SP responds with a list of the available target types, e.g.:

```
type_id="Public", type_name="Internet Gateway"
type_id="VPN 1", type_name="VPN Gateway", protocol="IPSec",
  version="1"
type_id="VM Resource", type_name="Virtual Machines"
```

Note: The above are not actual entities, they are types of targets which the SP supports in the context of this tenant.

The tenant makes another call to request creating a target corresponding to one of these types - e.g.

```
create_target(tenantID, name="My Internet Gateway", type_id="Public").
```

The SP creates a target resource (an actual entity) specific to this tenant (and also the necessary configuration to instantiate the "Internet Gateway" in the case of the above example).

The tenant can refer to this target in multiple routes (and/or create additional such targets).

Creating a Target to point to a Resource with an IP address

There are two possibilities here:

1. The resource is owned by the same tenant

   The tenant creates a target with the information about the VM - e.g.

   ```
   create_target(tenantID, name="My NAT", type_id="VM
   Resource", resource_id="nat_vm_uuid",
   ip_addr="vm_ip_address")
   ```

2. The resource is owned by a different tenant. There are again two possibilities here:

   a. The resource has an IP address on the SP's "public" network, e.g. a storage service

   ```
   create_target(tenantID, name="External Storage", type_id="VM Resource",
      ip_addr="storage_ip_address")
   ```

   b. The resource is on a private network,

   ```
   create_target(tenantID, name="target_web_server_x", type_id="VM
      Resource", target_tenant="web_server_tenant_uuid"
      ip_addr="server_x_private_ip",
      auth_token="token_to_access_web_server")
   ```

APIs

API at a glance

Resource	URL	GET	POST	PUT	DELETE
Subnets	/v2.0/tenants/{tenant-id}/subnets	Y	Y	-	-
	/v2.0/tenants/{tenant-id}/subnets/{subnet-id}	Y	-	Y	Y
Route-tables	/v2.0/tenants/{tenant-id}/route_table	Y	~~Y~~	Y	-
	~~/v2.0/tenants/{tenant-id}/route_tables/{routetable-id}~~	~~Y~~	~~-~~	~~Y~~	~~Y~~
Targets	/v2.0/tenants/{tenant-id}/targets	Y	Y	-	-
	/v2.0/tenants/{tenant-id}/targets/{target-id}	Y	-	Y	Y
	/v2.0/tenants/{tenant-id}/targets/types	Y			

[Strickthrough text – These API calls are not required with one route-table per tenant]

Subnets

```
def create_subnet(tenant_id, CIDR, **kwargs)
def show_subnet(tenant_id, subnet_id, **kwargs)
def update_subnet(tenant_id, subnet_id, **kwargs)
def delete_subnet(tenant_id, subnet_id, **kwargs)
def list_subnets(tenant_id, **kwargs))

kwargs:
name
description
network_id_list
```

Route-tables

~~create_route_table(tenant_id, routes_list, **kwargs)~~

show_route_table(tenant_id, ~~route_table_id,~~ **kwargs)

~~delete_route_table(tenant_id, route_table_id, **kwargs)~~

update_route_table(tenant_id, ~~route_table_id,~~ routes_list, **kwargs)

~~list_route_tables(tenant_id, **kwargs)~~

route_list:

List of routes in the format: Source, Destination, Target

kwargs:

name

description

[Strickthrough text – These API calls are not required with one route-table per tenant]

Targets

```
create_target(tenant_id, **kwargs)
kwargs:
name
description
type_id
resource_id
ip_address
target_tenant
auth_token

show_target(tenant_id, target_id, **kwargs)
delete_target(tenant_id, target_id, **kwargs)
update_target(tenant_id, target_id, **kwargs)

list_target_types(tenant_id, **kwargs)
```

Use Cases

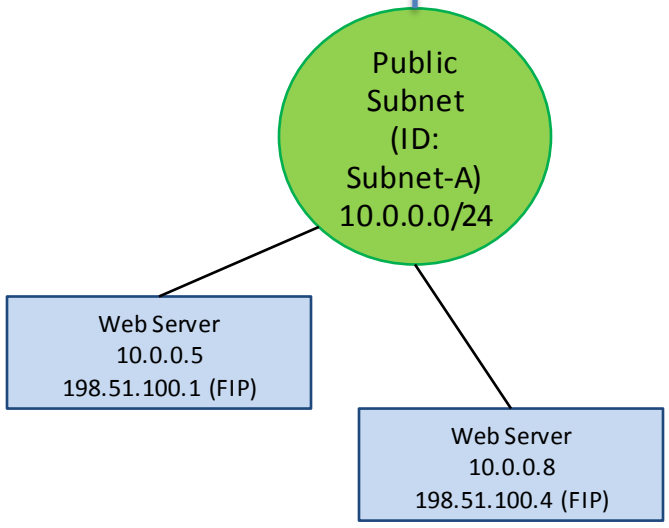
Conventions used in the following examples

- Private Subnets
  - These subnets reside on the tenant's private IP address space and do not have a route to the public network (Internet).
- Public Subnets
  - VMs on these subnets have a default route to the public network (Internet). They are also associated with Floating (public) IPs, hence they can be accessed from the Internet. This association of Floating IPs is a separate feature, and it's specification is not captured in this proposal.
- The following are not prescriptive, could be target type driven):
  - Intra-subnet traffic is always allowed, i.e. entities on the same subnet are always assumed to be directly connected and do not require an entry in the route-table.
  - Inter-subnet traffic is not allowed by default, i.e. there is no implicit routing to/from subnets; routing will happen between entities on different subnets only if a routing entry exists in the route-table.
  - Routes are reflexive, i.e. no explicit route entry needs to be created for a connection's return path. A stateful entity which tracks connection states might be required to achieve this.

Use case: One publicly accessible subnet

Public subnet can also initiate connections to the Internet

Source	Destination	Target
Subnet-A	0.0.0.0	“My Internet Gateway”*



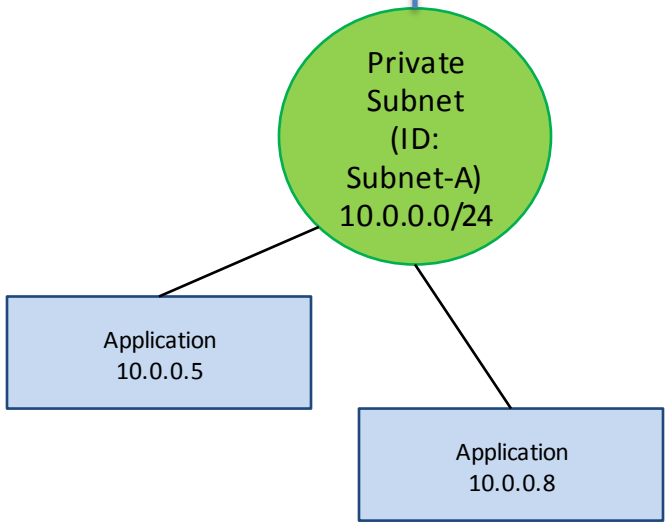
*Refers to the ID of a target resource created by the tenant with the following call:

```
create_target(tenantID, name="My Internet Gateway",
type_id="Public")
```

Use case: One private subnet

Private subnet can also initiate connections to the home network via VPN

Source	Destination	Target
Subnet-A	0.0.0.0	“My VPN Gateway”*



*Refers to the ID of a target resource created by the tenant with the following call:

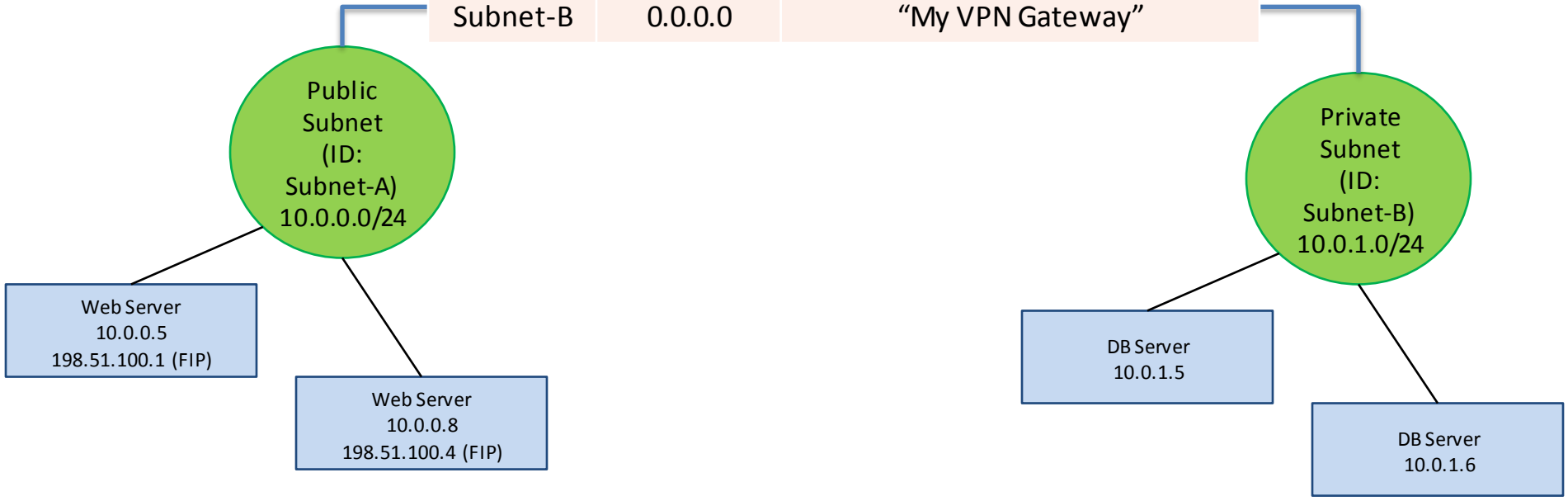
```
create_target(tenantID, name="My VPN Gateway", type_id="VPN 1", protocol="IPSec", version="1")
```

Use case: Public and VPN Access

One publicly accessible subnet, which can also initiate connections to the Internet

One private subnet, which can also initiate connections to the home network via VPN

Source	Destination	Target
Subnet-A	10.0.1.0/24	Subnet-B
Subnet-B	10.0.0.0/24	Subnet-A
Subnet-A	0.0.0.0	“My Internet Gateway”
Subnet-B	0.0.0.0	“My VPN Gateway”



Use case: 3-tier app

One publicly accessible subnet and two private Subnets

Public subnet can also initiate connections to the Internet