The NSX Terraform Provider

The NSX Terraform provider gives the NSX administrator a way to automate NSX to provide virtualized networking and security services using both ESXi and KVM based hypervisor hosts as well as container networking and security.

More information on NSX can be found on the NSX Product Page (https://www.vmware.com/products/nsx.html)

Documentation on the NSX platform can be found on the NSX Documentation Page (https://docs.vmware.com/en/VMware-NSX-T/index.html)

Please use the navigation to the left to read about available data sources and resources.

Basic Configuration of the NSX Terraform Provider

In order to use the NSX Terraform provider you must first configure the provider to communicate with the VMware NSX manager. The NSX manager is the system which serves the NSX REST API and provides a way to configure the desired state of the NSX system. The configuration of the NSX provider requires the IP address, hostname, or FQDN of the NSX manager.

The NSX provider offers several ways to authenticate to the NSX manager. Credentials can be provided statically or provided as environment variables. In addition, client certificates can be used for authentication. For authentication with certificates Terraform will require a certificate file and private key file in PEM format. To use client certificates the client certificate needs to be registered with NSX-T manager prior to invoking Terraform.

The provider also can accept both signed and self-signed server certificates. It is recommended that in production environments you only use certificates signed by a certificate authority. NSX ships by default with a self-signed server certificates as the hostname of the NSX manager is not known until the NSX administrator determines what name or IP to use.

Setting the `allow_unverified_ssl` parameter to `true` will direct the Terraform client to skip server certificate verification. This is not recommended in production deployments as it is recommended that you use trusted connection using certificates signed by a certificate authority.

With the `ca_file` parameter you can also specify a file that contains your certificate authority certificate in PEM format to verify certificates with a certificate authority.

There are also a number of other parameters that can be set to tune how the provider connects to the NSX REST API. It is recommended you leave these to the defaults unless you experience issues in which case they can be tuned to optimize the system in your environment.

Note that in all of the examples you will need to update the `host`, `username`, and `password` settings to match those configured in your NSX deployment.

Example of Configuration with Credentials

```
provider "nsxt" {
  host                     = "192.168.110.41"
  username                 = "admin"
  password                 = "default"
  allow_unverified_ssl     = true
  max_retries              = 10
  retry_min_delay          = 500
  retry_max_delay          = 5000
  retry_on_status_codes    = [429]
}
```

Example of Setting Environment Variables

```
export NSXT_MANAGER_HOST="192.168.110.41"
export NSXT_USERNAME="admin"
export NSXT_PASSWORD="default"
```

Example using a Client Certificate

```
provider "nsxt" {
  host                  = "192.168.110.41"
  client_auth_cert_file = "mycert.pem"
  client_auth_key_file  = "mykey.pem"
  allow_unverified_ssl  = true
}
```

Example with Certificate Authority Certificate

```
provider "nsxt" {
  host     = "10.160.94.11"
  username = "admin"
  password = "qwerty"
  ca_file  = "myca.pem"
}
```

Argument Reference

The following arguments are used to configure the VMware NSX-T Provider:

- `host` - (Required) The host name or IP address of the NSX-T manager. Can also be specified with the `NSXT_MANAGER_HOST` environment variable.
- `username` - (Required) The user name to connect to the NSX-T manager as. Can also be specified with the `NSXT_USERNAME` environment variable.

- `password` - (Required) The password for the NSX-T manager user. Can also be specified with the `NSXT_PASSWORD` environment variable.
- `client_auth_cert_file` - (Optional) The path to a certificate file for certificate authorization. Can also be specified with the `NSXT_CLIENT_AUTH_CERT_FILE` environment variable.
- `client_auth_key_file` - (Optional) The path to a private key file for the certificate supplied to `client_auth_cert_file`. Can also be specified with the `NSXT_CLIENT_AUTH_KEY_FILE` environment variable.
- `allow_unverified_ssl` - (Optional) Boolean that can be set to true to disable SSL certificate verification. This should be used with care as it could allow an attacker to intercept your auth token. If omitted, default value is `false`. Can also be specified with the `NSXT_ALLOW_UNVERIFIED_SSL` environment variable.
- `ca_file` - (Optional) The path to an optional CA certificate file for SSL validation. Can also be specified with the `NSXT_CA_FILE` environment variable.
- `max_retries` - (Optional) The maximum number of retires before failing an API request. Default: `10` Can also be specified with the `NSXT_MAX_RETRIES` environment variable.
- `retry_min_delay` - (Optional) The minimum delay, in milliseconds, between retires made to the API. Default: `500`. Can also be specified with the `NSXT_RETRY_MIN_DELAY` environment variable.
- `retry_max_delay` - (Optional) The maximum delay, in milliseconds, between retires made to the API. Default: `5000`. Can also be specified with the `NSXT_RETRY_MAX_DELAY` environment variable.
- `retry_on_status_codes` - (Optional) A list of HTTP status codes to retry on. By default, the provider will retry on HTTP error 429 (too many requests), essentially retrying on throttled connections. Can also be specified with the `NSXT_RETRY_ON_STATUS_CODES` environment variable.

NSX Logical Networking

The NSX Terraform provider can be used to manage logical networking and security constructs in NSX. This includes logical switching, routing and firewall.

Logical Networking and Security Example Usage

The following example demonstrates using the NSX Terraform provider to create a logical switch and tier1 logical router. It then connects the logical switch to the tier1 logical router and uplinks the T1 router to a pre-created T0 router.

Example variables.tf File

This file allows you to define some variables that can be reused in multiple .tf files.

```
variable "nsx_manager" {}
variable "nsx_username" {}
variable "nsx_password" {}
```

Example terraform.tfvars File

This file allows you to set some variables that can be reused in multiple .tf files.

```
nsx_manager = "192.168.110.41"

nsx_username = "admin"

nsx_password = "default"
```

Example nsx.tf file

This file will define the logical networking topology that Terraform will create in NSX.

```
#
# The first step is to configure the VMware NSX provider to connect to the NSX
# REST API running on the NSX manager.
#
provider "nsxt" {
  host                  = "${var.nsx_manager}"
  username              = "${var.nsx_username}"
  password              = "${var.nsx_password}"
  allow_unverified_ssl  = true
  max_retries           = 10
  retry_min_delay       = 500
  retry_max_delay       = 5000
  retry_on_status_codes = [429]
}

#
# Here we show that you define a NSX tag which can be used later to easily to
# search for the created objects in NSX.
#
variable "nsx_tag_scope" {
  default = "project"
}

variable "nsx_tag" {
  default = "terraform-demo"
}

#
# This part of the example shows some data sources we will need to refer to
# later in the .tf file. They include the transport zone, tier 0 router and
# edge cluster.
#
data "nsxt_transport_zone" "overlay_tz" {
  display_name = "tz1"
}

#
# The tier 0 router (T0) is considered a "provider" router that is pre-created
# by the NSX admin. A T0 router is used for north/south connectivity between
# the logical networking space and the physical networking space. Many tier 1
# routers will be connected to a tier 0 router.
#
data "nsxt_logical_tier0_router" "tier0_router" {
  display_name = "DefaultT0Router"
}

data "nsxt_edge_cluster" "edge_cluster1" {
  display_name = "EdgeCluster1"
}
```

```
}

#
# This shows the settings required to create a NSX logical switch to which you
# can attach virtual machines.
#
resource "nsxt_logical_switch" "switch1" {
  admin_state       = "UP"
  description       = "LS created by Terraform"
  display_name      = "TfLogicalSwitch"
  transport_zone_id = "${data.nsxt_transport_zone.overlay_tz.id}"
  replication_mode  = "MTEP"

  tag {
    scope = "${var.nsx_tag_scope}"
    tag   = "${var.nsx_tag}"
  }

  tag {
    scope = "tenant"
    tag   = "second_example_tag"
  }
}

#
# In this part of the example the settings are defined that are required to
# create a T1 router. In NSX a T1 router is often used on a per user, tenant,
# or application basis. Each application may have it's own T1 router. The T1
# router provides the default gateway for machines on logical switches
# connected to the T1 router.
#
resource "nsxt_logical_tier1_router" "tier1_router" {
  description                 = "Tier1 router provisioned by Terraform"
  display_name                = "TfTier1"
  failover_mode               = "PREEMPTIVE"
  high_availability_mode      = "ACTIVE_STANDBY"
  edge_cluster_id             = "${data.nsxt_edge_cluster.edge_cluster1.id}"
  enable_router_advertisement = true
  advertise_connected_routes  = true
  advertise_static_routes     = false
  advertise_nat_routes        = true

  tag {
    scope = "${var.nsx_tag_scope}"
    tag   = "${var.nsx_tag}"
  }
}

#
# This resource creates a logical port on the T0 router. We will connect the T1
# router to this port to enable connectivity from the tenant / application
# networks to the networks to the cloud.
#
resource "nsxt_logical_router_link_port_on_tier0" "link_port_tier0" {
  description       = "TIER0_PORT1 provisioned by Terraform"
  display_name      = "TIER0_PORT1"
  logical_router_id = "${data.nsxt_logical_tier0_router.tier0_router.id}"

  tag {
    scope = "${var.nsx_tag_scope}"
    tag   = "${var.nsx_tag}"
  }
}

#
# Here we create a tier 1 router uplink port and connect it to T0 router port
```

```
# created in the previous example.
#
resource "nsxt_logical_router_link_port_on_tier1" "link_port_tier1" {
  description                   = "TIER1_PORT1 provisioned by Terraform"
  display_name                  = "TIER1_PORT1"
  logical_router_id             = "${nsxt_logical_tier1_router.tier1_router.id}"
  linked_logical_router_port_id = "${nsxt_logical_router_link_port_on_tier0.link_port_tier0.id}"

  tag {
    scope = "${var.nsx_tag_scope}"
    tag   = "${var.nsx_tag}"
  }
}

#
# Like their physical counterpart a logical switch can have switch ports. In
# this example Terraform will create a logical switch port on a logical switch.
#
resource "nsxt_logical_port" "logical_port1" {
  admin_state       = "UP"
  description       = "LP1 provisioned by Terraform"
  display_name      = "LP1"
  logical_switch_id = "${nsxt_logical_switch.switch1.id}"

  tag {
    scope = "${var.nsx_tag_scope}"
    tag   = "${var.nsx_tag}"
  }
}

#
# In order to connect a logical switch to a tier 1 logical router we will need
# a downlink port on the tier 1 router and will need to  connect it to the
# switch port we created above.
#
# The IP address provided in the `ip_address` property will be default gateway
# for virtual machines connected to this logical switch.
#
resource "nsxt_logical_router_downlink_port" "downlink_port" {
  description                   = "DP1 provisioned by Terraform"
  display_name                  = "DP1"
  logical_router_id             = "${nsxt_logical_tier1_router.tier1_router.id}"
  linked_logical_switch_port_id = "${nsxt_logical_port.logical_port1.id}"
  ip_address                    = "192.168.245.1/24"

  tag {
    scope = "${var.nsx_tag_scope}"
    tag   = "${var.nsx_tag}"
  }
}
```

In order to be able to connect VMs to the newly created logical switch a new `vpshere_network` datasource need to be defined.

```
data "vsphere_network" "terraform_switch1" {
    name = "${nsxt_logical_switch.switch1.display_name}"
    datacenter_id = "${data.vsphere_datacenter.dc.id}"
    depends_on = ["nsxt_logical_switch.switch1"]
}
```

The datasource in the above example should be referred in `network_id` inside `network_interface` section for

`vsphere_virtual_machine` resource.

Feature Requests, Bug Reports, and Contributing

For more information how how to submit feature requests, bug reports, or details on how to make your own contributions to the provider, see the NSX-T provider project page (https://github.com/terraform-providers/terraform-provider-nsxt).

nsxt_edge_cluster

This data source provides information about Edge clusters configured in NSX. An Edge cluster is a collection of Edge nodes which can be deployed as either VM form-factor or bare-metal form-factor machines for connectivity between overlay logical switches and non-NSX underlay networking for north/south layer 2 or layer 3 connectivity. Each T0 router will be placed on one ore more Edge nodes in an Edge cluster therefore this data source is needed for the creation of T0 logical routers.

Example Usage

```
data "nsxt_edge_cluster" "edge_cluster1" {
  display_name = "edgecluster"
}
```

Argument Reference

- `id` - (Optional) The ID of Edge Cluster to retrieve.
- `display_name` - (Optional) The Display Name prefix of the Edge Cluster to retrieve.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `description` - The description of the edge cluster.
- `deployment_type` - This field could show deployment_type of members. It would return UNKNOWN if there is no members, and return VIRTUAL_MACHINE|PHYSICAL_MACHINE if all Edge members are VIRTUAL_MACHINE|PHYSICAL_MACHINE.
- `member_node_type` - An Edge cluster is homogeneous collection of NSX transport nodes used for north/south connectivity between NSX logical networking and physical networking. Hence all transport nodes of the cluster must be of same type. This field shows the type of transport node,

nsxt_logical_tier0_router

This data source provides information about logical Tier 0 routers configured in NSX. A Tier 0 router is used to connect NSX networking with traditional physical networking. Tier 0 routers are placed on an Edge cluster and will exist on one or more Edge node depending on deployment settings (i.e. active/active or active/passive). A Tier 0 router forwards layer 3 IP packets and typically peers with a traditional physical router using BGP or can use static routing.

Example Usage

```
data "nsxt_logical_tier0_router" "tier0_router" {
  display_name = "PLR1"
}
```

Argument Reference

- `id` - (Optional) The ID of Logical Tier 0 Router to retrieve.
- `display_name` - (Optional) The Display Name prefix of the Logical Tier 0 Router to retrieve.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `description` - The description of the logical Tier 0 router.
- `edge_cluster_id` - The id of the Edge cluster where this logical router is placed.
- `high_availability_mode` - The high availability mode of this logical router.

nsxt_ns_service

This data source provides information about a network and security (NS) service configured in NSX. NS services are either factory defined in NSX or can be defined by the NSX administrator. They provide a convenience name for a port/protocol pair that is often used in fire walling or load balancing.

Example Usage

```
data "nsxt_ns_service" "ns_service_dns" {
  display_name = "DNS"
}
```

Argument Reference

- `id` - (Optional) The ID of NS service to retrieve
- `display_name` - (Optional) The Display Name of the NS service to retrieve.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `description` - The description of the NS service.

nsxt_switching_profile

The switching profile data source provides information about switching profiles configured in NSX. A switching profile is a template that defines the settings of one or more logical switches. There can be both factory default and user defined switching profiles. One example of a switching profile is a quality of service (QoS) profile which defines the QoS settings of all switches that use the defined switch profile.

Example Usage

```
data "nsxt_switching_profile" "qos_profile" {
  display_name = "qos-profile"
}
```

Argument Reference

- `id` - (Optional) The ID of Switching Profile to retrieve.
- `display_name` - (Optional) The Display Name of the Switching Profile to retrieve.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `resource_type` - The resource type representing the specific type of this switching profile.
- `description` - The description of the switching profile.

nsxt_transport_zone

This data source provides information about Transport Zones (TZ) configured in NSX. A Transport Zone defines the scope to which a network can extend in NSX. For example an overlay based Transport Zone is associated with both hypervisors and logical switches and defines which hypervisors will be able to serve the defined logical switch. Virtual machines on the hypervisor associated with a Transport Zone can be attached to logical switches in that same Transport Zone.

Example Usage

```
data "nsxt_transport_zone" "overlay_transport_zone" {
  display_name = "1-transportzone-87"
}
```

Argument Reference

- `id` - (Optional) The ID of Transport Zone to retrieve.
- `display_name` - (Optional) The Display Name prefix of the Transport Zone to retrieve.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `description` - The description of the Transport Zone.
- `host_switch_name` - The name of the N-VDS (host switch) on all Transport Nodes in this Transport Zone that will be used to run NSX network traffic.
- `transport_type` - The transport type of this transport zone (OVERLAY or VLAN).

nsxt_algorithm_type_ns_service

This resource provides a way to configure a networking and security service which can be used with the NSX firewall. A networking and security service is an object that contains the TCP/UDP algorithm, source ports and destination ports in a single entity.

Example Usage

```
resource "nsxt_algorithm_type_ns_service" "ns_service_alg" {
  description       = "S1 provisioned by Terraform"
  display_name      = "S1"
  algorithm         = "FTP"
  destination_port  = "21"
  source_ports      = ["9001-9003"]

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description.
- `destination_port` - (Required) a single destination port.
- `source_ports` - (Optional) Set of source ports/ranges.
- `algorithm` - (Required) Algorithm one of "ORACLE_TNS", "FTP", "SUN_RPC_TCP", "SUN_RPC_UDP", "MS_RPC_TCP", "MS_RPC_UDP", "NBNS_BROADCAST", "NBDG_BROADCAST", "TFTP"
- `tag` - (Optional) A list of scope + tag pairs to associate with this service.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `default_service` - The default NSServices are created in the system by default. These NSServices can't be modified/deleted.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing Algorithm type NS service can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_algorithm_type_ns_service.ns_service_alg UUID
```

The above command imports the algorithm based networking and security service named `ns_service_alg` with the NSX id `UUID`.

nsxt_dhcp_relay_profile

This resource can be used to configure a NSX DHCP relay profile on the NSX manager. A DHCP relay profile is a type of template that can be used to define a remote DHCP server where DHCP packets can be relayed for DHCP requests of machines attached to NSX logical topologies. The DHCP relay profile can be used in a DHCP relay service and later consumed by a router link port.

Example Usage

```
resource "nsxt_dhcp_relay_profile" "dr_profile" {
  description  = "DRP provisioned by Terraform"
  display_name = "DRP"

  tag {
    scope = "color"
    tag   = "red"
  }

  server_addresses = ["1.1.1.1"]
}

resource "nsxt_dhcp_relay_service" "dr_service" {
  display_name          = "DRS"
  dhcp_relay_profile_id = "${nsxt_dhcp_relay_profile.dr_profile.id}"
}

resource "nsxt_logical_router_downlink_port" "router_downlink" {
  display_name                  = "logical_router_downlink_port"
  linked_logical_switch_port_id = "${nsxt_logical_port.port1.id}"
  logical_router_id             = "${nsxt_logical_tier1_router.rtr1.id}"

  subnet {
    ip_addresses  = ["8.0.0.1"]
    prefix_length = 24
  }

  service_binding {
    target_id   = "${nsxt_dhcp_relay_service.dr_service.id}"
    target_type = "LogicalService"
  }
}
```

Argument Reference

The following arguments are supported:

- `description` - (Optional) Description of this resource.
- `display_name` - (Optional) The display name of this resource. Defaults to ID if not set.
- `tag` - (Optional) A list of scope + tag pairs to associate with this dhcp relay profile.
- `server_addresses` - (Required) IP addresses of the DHCP relay servers.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the dhcp_relay_profile.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing DHCP Relay profile can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_dhcp_relay_profile.dr_profile UUID
```

The above command imports the DHCP relay profile named `dr_profile` with the NSX id `UUID`.

nsxt_dhcp_relay_service

This resource provides a way to configure the DHCP relay service on the NSX manager. The DHCP relay service uses a DHCP relay profile and later consumed by a router link port to provide DHCP addresses to virtual machines connected to a logical switch.

Example Usage

```
resource "nsxt_dhcp_relay_profile" "dr_profile" {
  description  = "DRP provisioned by Terraform"
  display_name = "DRP"

  tag {
    scope = "color"
    tag   = "red"
  }

  server_addresses = ["1.1.1.1"]
}

resource "nsxt_dhcp_relay_service" "dr_service" {
  display_name          = "DRS"
  dhcp_relay_profile_id = "${nsxt_dhcp_relay_profile.dr_profile.id}"
}

resource "nsxt_logical_router_downlink_port" "router_downlink" {
  display_name                  = "logical_router_downlink_port"
  linked_logical_switch_port_id = "${nsxt_logical_port.port1.id}"
  logical_router_id             = "${nsxt_logical_tier1_router.rtr1.id}"

  subnet {
    ip_addresses  = ["8.0.0.1"]
    prefix_length = 24
  }

  service_binding {
    target_id   = "${nsxt_dhcp_relay_service.dr_service.id}"
    target_type = "LogicalService"
  }
}
```

Argument Reference

The following arguments are supported:

- `description` - (Optional) Description of this resource.
- `display_name` - (Optional) The display name of this resource. Defaults to ID if not set.
- `tag` - (Optional) A list of scope + tag pairs to associate with this dhcp_relay_service.
- `dhcp_relay_profile_id` - (Required) dhcp relay profile referenced by the dhcp relay service.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the dhcp_relay_service.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing DHCP Relay service can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_dhcp_relay_service.dr_service UUID
```

The above command imports the DHCP relay service named `dr_service` with the NSX id `UUID`.

nsxt_ether_type_ns_service

This resource provides a way to configure a networking and security service which can be used within NSX. This specific service is for the layer 2 Ethernet protocol.

Example Usage

```
resource "nsxt_ether_type_ns_service" "etns" {
  description  = "S1 provisioned by Terraform"
  display_name = "S1"
  ether_type   = "1536"

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description.
- `ether_type` - (Required) Type of the encapsulated protocol.
- `tag` - (Optional) A list of scope + tag pairs to associate with this service.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `default_service` - The default NSServices are created in the system by default. These NSServices can't be modified/deleted.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing Ethernet type NS service can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_ether_type_ns_service.etns UUID
```

The above command imports the ethernet type networking and security service named `etns` with the NSX id `UUID`.

nsxt_firewall_section

This resource provides a way to configure a firewall section on the NSX manager. A firewall section is a collection of firewall rules that are grouped together.

Example Usage

```
resource "nsxt_firewall_section" "firewall_sect" {
  description  = "FS provisioned by Terraform"
  display_name = "FS"

  tag {
    scope = "color"
    tag   = "blue"
  }

  applied_to {
    target_type = "NSGroup"
    target_id   = "${nsxt_ns_group.group1.id}"
  }

  section_type = "LAYER3"
  stateful     = true

  rule {
    display_name = "out_rule"
    description  = "Out going rule"
    action       = "ALLOW"
    logged       = true
    ip_protocol  = "IPV4"
    direction    = "OUT"

    source {
      target_type = "LogicalSwitch"
      target_id   = "${nsxt_logical_switch.switch1.id}"
    }

    destination {
      target_type = "LogicalSwitch"
      target_id   = "${nsxt_logical_switch.switch2.id}"
    }
  }

  rule {
    display_name = "in_rule"
    description  = "In going rule"
    action       = "DROP"
    logged       = true
    ip_protocol  = "IPV4"
    direction    = "IN"

    service {
      target_type = "NSService"
      target_id   = "e8d59e13-484b-4825-ae3b-4c11f83249d9"
    }

    service {
      target_type = "NSService"
      target_id   = "${nsxt_l4_port_set_ns_service.http.id}"
    }
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) The display name of this firewall section. Defaults to ID if not set.
- `description` - (Optional) Description of this firewall section.
- `tag` - (Optional) A list of scope + tag pairs to associate with this firewall section.
- `applied_to` - (Optional) List of objects where the rules in this section will be enforced. This will take precedence over rule level applied_to. [Supported target types: "LogicalPort", "LogicalSwitch", "NSGroup"]
- `section_type` - (Required) Type of the rules which a section can contain. Either LAYER2 or LAYER3. Only homogeneous sections are supported.
- `stateful` - (Required) Stateful or Stateless nature of firewall section is enforced on all rules inside the section. Layer3 sections can be stateful or stateless. Layer2 sections can only be stateless.
- `rule` - (Optional) A list of rules to be applied in this section. each rule has the following arguments:
  - `display_name` - (Optional) The display name of this rule. Defaults to ID if not set.
  - `description` - (Optional) Description of this rule.
  - `action` - (Required) Action enforced on the packets which matches the firewall rule. [Allowed values: "ALLOW", "DROP", "REJECT"]
  - `applied_to` - (Optional) List of object where rule will be enforced. The section level field overrides this one. Null will be treated as any. [Supported target types: "LogicalPort", "LogicalSwitch", "NSGroup"]
  - `destination` - (Optional) List of the destinations. Null will be treated as any. [Allowed target types: "IPSet", "LogicalPort", "LogicalSwitch", "NSGroup", "MACSet" (depending on the section type)]
  - `destinations_excluded` - (Optional) Negation of the destination.
  - `direction` - (Optional) Rule direction in case of stateless firewall rules. This will only considered if section level parameter is set to stateless. Default to IN_OUT if not specified. [Allowed values: "IN", "OUT", "IN_OUT"]
  - `disabled` - (Optional) Flag to disable rule. Disabled will only be persisted but never provisioned/realized.
  - `ip_protocol` - (Optional) Type of IP packet that should be matched while enforcing the rule. [allowed values: "IPV4", "IPV6", "IPV4_IPV6"]
  - `logged` - (Optional) Flag to enable packet logging. Default is disabled.
  - `notes` - (Optional) User notes specific to the rule.
  - `rule_tag` - (Optional) User level field which will be printed in CLI and packet logs.
  - `service` - (Optional) List of the services. Null will be treated as any. [Allowed target types: "NSService", "NSServiceGroup"]
  - `source` - (Optional) List of sources. Null will be treated as any. [Allowed target types: "IPSet", "LogicalPort", "LogicalSwitch", "NSGroup", "MACSet" (depending on the section type)]
  - `sources_excluded` - (Optional) Negation of the source.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the firewall_section.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.
- `is_default` - A boolean flag which reflects whether a firewall section is default section or not. Each Layer 3 and Layer 2 section will have at least and at most one default section.

Importing

An existing Firewall section can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_firewall_section.firewall_sect UUID
```

The above command imports the firewall section named `firewall_sect` with the NSX id `UUID`.

nsxt_icmp_type_ns_service

This resource provides a way to configure a networking and security service which can be used within NSX. This specific service is for the ICMP protocol.

Example Usage

```
resource "nsxt_icmp_type_ns_service" "ns_service_icmp" {
  description  = "S1 provisioned by Terraform"
  display_name = "S1"
  protocol     = "ICMPv4"
  icmp_type    = "5"
  icmp_code    = "1"

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description.
- `protocol` - (Required) Version of ICMP protocol ICMPv4 or ICMPv6.
- `icmp_type` - (Optional) ICMP message type.
- `icmp_code` - (Optional) ICMP message code
- `tag` - (Optional) A list of scope + tag pairs to associate with this service.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `default_service` - The default NSServices are created in the system by default. These NSServices can't be modified/deleted.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing ICMP type NS Service can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_icmp_type_ns_service.x id
```

The above service imports the ICMP type network and security service named `x` with the NSX id `id`.

nsxt_igmp_type_ns_service

This resource provides a way to configure a networking and security service which can be used within NSX. This specific service is for the IGMP protocol.

Example Usage

```
resource "nsxt_igmp_type_ns_service" "ns_service_igmp" {
  description  = "S1 provisioned by Terraform"
  display_name = "S1"

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description.
- `tag` - (Optional) A list of scope + tag pairs to associate with this service.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `default_service` - The default NSServices are created in the system by default. These NSServices can't be modified/deleted.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing IGMP type NS Service can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_igmp_type_ns_service.ns_service_igmp UUID
```

The above command imports the IGMP based networking and security service named `ns_service_igmp` with the NSX id `UUID`.

nsxt_ip_protocol_ns_service

This resource provides a way to configure a networking and security service which can be used within NSX. This specific service is for the IP protocol.

Example Usage

```
resource "nsxt_ip_protocol_ns_service" "ns_service_ip" {
  description  = "S1 provisioned by Terraform"
  display_name = "S1"
  protocol     = "10"

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description.
- `protocol` - (Required) IP protocol number (0-255)
- `tag` - (Optional) A list of scope + tag pairs to associate with this service.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `default_service` - The default NSServices are created in the system by default. These NSServices can't be modified/deleted.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing IP protocol NS service can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_ip_protocol_ns_service.ns_service_ip UUID
```

The above command imports the IP protocol based networking and security service named `ns_service_ip` with the NSX id `UUID`.

nsxt_ip_set

This resources provides a way to configure an IP set in NSX. An IP set is a collection of IP addresses. It is often used in the configuration of the NSX firewall.

Example Usage

```
resource "nsxt_ip_set" "ip_set1" {
  description  = "IS provisioned by Terraform"
  display_name = "IS"

  tag {
    scope = "color"
    tag   = "blue"
  }

  ip_addresses = ["1.1.1.1", "2.2.2.2"]
}
```

Argument Reference

The following arguments are supported:

- `description` - (Optional) Description of this resource.
- `display_name` - (Optional) The display name of this resource. Defaults to ID if not set.
- `tag` - (Optional) A list of scope + tag pairs to associate with this ip_set.
- `ip_addresses` - (Optional) IP addresses.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the ip_set.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing IP set can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_ip_set.ip_set1 UUID
```

The above command imports the IP set named `ip_set1` with the NSX id `UUID`.

nsxt_l4_port_set_ns_service

This resource provides a way to configure a networking and security service which can be used within NSX. This specific service is for configuration of layer 4 ports.

Example Usage

```
resource "nsxt_l4_port_set_ns_service" "ns_service_l4" {
  description       = "S1 provisioned by Terraform"
  display_name      = "S1"
  protocol          = "TCP"
  destination_ports = ["73", "8080", "81"]

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description of this resource.
- `destination_ports` - (Optional) Set of destination ports.
- `source_ports` - (Optional) Set of source ports.
- `protocol` - (Optional) VL4 protocol
- `tag` - (Optional) A list of scope + tag pairs to associate with this service.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `default_service` - The default NSServices are created in the system by default. These NSServices can't be modified/deleted.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing L4 port set NS service can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_l4_port_set_ns_service.ns_service_l4 UUID
```

The above command imports the layer 4 port based networking and security service named `ns_service_l4` with the NSX id `UUID`.

nsxt_logical_port

This resource provides a resource to configure a logical port on a logical switch in the NSX system. Like physical switches a logical switch can have one or more ports which can be connected to virtual machines or logical routers.

Example Usage

```
resource "nsxt_logical_port" "logical_port" {
  admin_state       = "UP"
  description       = "LP1 provisioned by Terraform"
  display_name      = "LP1"
  logical_switch_id = "${nsxt_logical_switch.switch1.id}"

  tag {
    scope = "color"
    tag   = "blue"
  }

  switching_profile_id {
    key   = "${data.nsxt_switching_profile.qos_profile.resource_type}"
    value = "${data.nsxt_switching_profile.qos_profile.id}"
  }
}
```

Argument Reference

The following arguments are supported:

- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description of this resource.
- `logical_switch_id` - (Required) Logical switch ID for the logical port.
- `admin_state` - (Optional) Admin state for the logical port. Accepted values - 'UP' or 'DOWN'. The default value is 'UP'.
- `switching_profile_id` - (Optional) List of IDs of switching profiles (of various types) to be associated with this switch. Default switching profiles will be used if not specified.
- `tag` - (Optional) A list of scope + tag pairs to associate with this logical port.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing Logical Port can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_logical_port.logical_port UUID
```

The above command imports the logical port named `logical_port` with the NSX id `UUID`.

nsxt_logical_router_downlink_port

This resource provides a means to define a downlink port on a logical router to connect a logical router to a logical switch. The result of this is to provide a default gateway to virtual machines running on the logical switch.

Example Usage

```
resource "nsxt_logical_router_downlink_port" "downlink_port" {
  description                   = "DP1 provisioned by Terraform"
  display_name                  = "DP1"
  logical_router_id             = "${nsxt_logical_router.rtr1.id}"
  linked_logical_switch_port_id = "${nsxt_logical_port.logical_port1.id}"
  ip_address                    = "1.1.0.1/24"

  service_binding {
    target_id   = "${nsxt_dhcp_relay_service.dr_service.id}"
    target_type = "LogicalService"
  }

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `logical_router_id` - (Required) Identifier for logical Tier-1 router on which this port is created
- `linked_logical_switch_port_id` - (Required) Identifier for port on logical switch to connect to
- `ip_address` - (Required) Logical router port subnet (ip_address / prefix length)
- `mac_address` - (Optional) Mac Address
- `urpf_mode` - (Optional) Unicast Reverse Path Forwarding mode
- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description of the resource.
- `tag` - (Optional) A list of scope + tag pairs to associate with this port.
- `service_binding` - (Optional) A list of services for this port

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.

- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing Logical Router Downlink Port can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_logical_router_downlink_port.downlink_port UUID
```

The above command imports the logical router downlink port named `downlink_port` with the NSX id `UUID`.

nsxt_logical_router_link_port_on_tier0

This resource provides the ability to configure a logical router link port on a tier 0 logical router. This port can then be used to connect the tier 0 logical router to another logical router.

Example Usage

```
resource "nsxt_logical_router_link_port_on_tier0" "link_port_tier0" {
  description       = "TIER0_PORT1 provisioned by Terraform"
  display_name      = "TIER0_PORT1"
  logical_router_id = "${data.nsxt_logical_tier0_router.rtr1.id}"

  service_binding {
    target_id   = "${nsxt_dhcp_relay_service.dr_service.id}"
    target_type = "LogicalService"
  }

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `logical_router_id` - (Required) Identifier for logical Tier0 router on which this port is created.
- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description of the resource.
- `tag` - (Optional) A list of scope + tag pairs to associate with this port.
- `service_binding` - (Optional) A list of services for this port

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `linked_logical_switch_port_id` - Identifier for port on logical router to connect to.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing logical router link port on Tier-0 can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_logical_router_link_port_on_tier0.link_port_tier0 UUID
```

The above command imports the logical router link port on the tier 0 logical router named `link_port_tier0` with the NSX id `UUID`.

nsxt_logical_router_link_port_on_tier1

This resource provides the ability to configure a logical router link port on a tier 1 logical router. This port can then be used to connect the tier 1 logical router to another logical router.

Example Usage

```
resource "nsxt_logical_router_link_port_on_tier1" "link_port_tier1" {
  description                   = "TIER1_PORT1 provisioned by Terraform"
  display_name                  = "TIER1_PORT1"
  logical_router_id             = "${nsxt_logical_tier1_router.rtr1.id}"
  linked_logical_router_port_id = "${nsxt_logical_router_link_port_on_tier0.link_port_tier0.id}"

  service_binding {
    target_id   = "${nsxt_dhcp_relay_service.dr_service.id}"
    target_type = "LogicalService"
  }

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `logical_router_id` - (Required) Identifier for logical tier-1 router on which this port is created.
- `linked_logical_switch_port_id` - (Required) Identifier for port on logical Tier-0 router to connect to.
- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description of the resource.
- `tag` - (Optional) A list of scope + tag pairs to associate with this port.
- `service_binding` - (Optional) A list of services for this port

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing logical router link port on Tier-1 can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_logical_router_link_port_on_tier1.link_port_tier1 UUID
```

The above command imports the logical router link port on the tier 1 router named `link_port_tier1` with the NSX id `UUID`.

nsxt_logical_switch

This resource provides a method to create a logical switch in NSX. Virtual machines can then be connected to the appropriate logical switch for the desired topology and network connectivity.

Example Usage

```
resource "nsxt_logical_switch" "switch1" {
  admin_state       = "UP"
  description       = "LS1 provisioned by Terraform"
  display_name      = "LS1"
  transport_zone_id = "${data.nsxt_transport_zone.transport_zone.id}"
  replication_mode  = "MTEP"

  tag {
    scope = "color"
    tag   = "blue"
  }

  switching_profile_id {
    key   = "${data.nsxt_switching_profile.qos_profiles.resource_type}"
    value = "${data.nsxt_switching_profile.qos_profiles.id}"
  }
}
```

Argument Reference

The following arguments are supported:

- `transport_zone_id` - (Required) Transport Zone ID for the logical switch.
- `admin_state` - (Optional) Admin state for the logical switch. Accepted values - 'UP' or 'DOWN'. The default value is 'UP'.
- `replication_mode` - (Optional) Replication mode of the Logical Switch. Accepted values - 'MTEP' (Hierarchical Two-Tier replication) and 'SOURCE' (Head Replication), with 'MTEP' being the default value. Applies to overlay logical switches.
- `switching_profile_id` - (Optional) List of IDs of switching profiles (of various types) to be associated with this switch. Default switching profiles will be used if not specified.
- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description of the resource.
- `ip_pool_id` - (Optional) Ip Pool ID to be associated with the logical switch.
- `mac_pool_id` - (Optional) Mac Pool ID to be associated with the logical switch.
- `vlan` - (Optional) Vlan for vlan logical switch. If not specified, this switch is overlay logical switch.
- `vni` - (Optional) Vni for the logical switch.
- `address_binding` - (Optional) List of Address Bindings for the logical switch. This setting allows to provide bindings between IP address, mac Address and vlan.

- `tag` - (Optional) A list of scope + tag pairs to associate with this logical switch.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical switch.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing X can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_logical_switch.switch1 UUID
```

The above command imports the logical switch named `switch1` with the NSX id `UUID`.

nsxt_logical_tier1_router

This resource provides a method for the management of a tier 1 logical router. A tier 1 logical router is often used for tenants, users and applications. There can be many tier 1 logical routers connected to a common tier 0 provider router.

Example Usage

```
resource "nsxt_logical_tier1_router" "tier1_router" {
  description                 = "RTR1 provisioned by Terraform"
  display_name                = "RTR1"
  failover_mode               = "PREEMPTIVE"
  edge_cluster_id             = "${data.nsxt_edge_cluster.edge_cluster.id}"
  enable_router_advertisement = true
  advertise_connected_routes  = false
  advertise_static_routes     = true
  advertise_nat_routes        = true

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `edge_cluster_id` - (Optional) Edge Cluster ID for the logical Tier1 router.
- `display_name` - (Optional) Display name, defaults to ID if not set.
- `description` - (Optional) Description of the resource.
- `tag` - (Optional) A list of scope + tag pairs to associate with this logical Tier1 router.
- `failover_mode` - (Optional) This failover mode determines, whether the preferred service router instance for given logical router will preempt the peer. Note - It can be specified if and only if logical router is ACTIVE_STANDBY and NON_PREEMPTIVE mode is supported only for a Tier1 logical router. For ACTIVE_ACTIVE logical routers, this field must not be populated
- `enable_router_advertisement` - (Optional) Enable the router advertisement
- `advertise_connected_routes` - (Optional) Enable the router advertisement for all NSX connected routes
- `advertise_static_routes` - (Optional) Enable the router advertisement for static routes
- `advertise_nat_routes` - (Optional) Enable the router advertisement for NAT routes

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the logical Tier1 router.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.
- `advertise_config_revision` - Indicates current revision number of the advertisement configuration object as seen by NSX-T API server. This attribute can be useful for debugging.
- `firewall_sections` - (Optional) The list of firewall sections for this router

Importing

An existing logical tier1 router can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_logical_tier1_router.tier1_router UUID
```

The above command imports the logical tier 1 router named `tier1_router` with the NSX id `UUID`.

nsxt_nat_rule

This resource provides a means to configure a NAT rule in NSX. NAT provides network address translation between one IP address space and another IP address space. NAT rules can be destination NAT or source NAT rules.

Example Usage

```
resource "nsxt_nat_rule" "rule1" {
  logical_router_id         = "${nsxt_logical_tier1_router.rtr1.id}"
  description               = "NR provisioned by Terraform"
  display_name              = "NR"
  action                    = "SNAT"
  enabled                   = true
  logging                   = true
  nat_pass                  = false
  translated_network        = "4.4.0.0/24"
  match_destination_network = "3.3.3.0/24"
  match_source_network      = "5.5.5.0/24"

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `logical_router_id` - (Required) ID of the logical router.
- `description` - (Optional) Description of this resource.
- `display_name` - (Optional) The display name of this resource. Defaults to ID if not set.
- `tag` - (Optional) A list of scope + tag pairs to associate with this NAT rule.
- `action` - (Required) NAT rule action type. Valid actions are: SNAT, DNAT, NO_NAT and REFLEXIVE. All rules in a logical router are either stateless or stateful. Mix is not supported. SNAT and DNAT are stateful, and can NOT be supported when the logical router is running at active-active HA mode. The REFLEXIVE action is stateless. The NO_NAT action has no translated_fields, only match fields.
- `enabled` - (Optional) enable/disable the rule.
- `logging` - (Optional) enable/disable the logging of rule.
- `match_destination_network` - (Optional) IP Address | CIDR | (null implies Any).
- `match_source_network` - (Optional) IP Address | CIDR | (null implies Any).
- `nat_pass` - (Optional) Enable/disable to bypass following firewall stage. The default is true, meaning that the following firewall stage will be skipped. Please note, if action is NO_NAT, then nat_pass must be set to true or omitted.

- `translated_network` - (Optional) IP Address | IP Range | CIDR. For DNAT rules only a single ip is supported.
- `translated_ports` - (Optional) port number or port range. DNAT only.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the nat_rule.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.
- `rule_priority` - The priority of the rule which is ascending, valid range [0-2147483647]. If multiple rules have the same priority, evaluation sequence is undefined.

Importing

An existing NAT rule can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_nat_rule.rule1 logical-router-uuid/nat-rule-num
```

The above command imports the NAT rule named `rule1` with the number id `nat-rule-num` that belongs to the tier 1 logical router with the NSX id `logical-router-uuid`.

nsxt_ns_group

This resource provides a method to create and manage a network and security (NS) group in NSX. A NS group is used to group other objects into collections for application of other settings.

Example Usage

```
resource "nsxt_ns_group" "group2" {
  description  = "NG provisioned by Terraform"
  display_name = "NG"

  member {
    target_type = "NSGroup"
    value       = "${nsxt_ns_group.group1.id}"
  }

  membership_criteria {
    target_type = "LogicalPort"
    scope       = "XXX"
    tag         = "YYY"
  }

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `description` - (Optional) Description of this resource.
- `display_name` - (Optional) The display name of this resource. Defaults to ID if not set.
- `tag` - (Optional) A list of scope + tag pairs to associate with this NS group.
- `member` - (Optional) Reference to the direct/static members of the NSGroup. Can be ID based expressions only. VirtualMachine cannot be added as a static member. target_type can be: NSGroup, IPSet, LogicalPort, LogicalSwitch, MACSet
- `membership_criteria` - (Optional) List of tag or ID expressions which define the membership criteria for this NSGroup. An object must satisfy at least one of these expressions to qualify as a member of this group.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the ns_group.

- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for debugging.

Importing

An existing networking and security group can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_ns_group.group2 UUID
```

The above command imports the networking and security group named `group2` with the NSX id `UUID`.

nsxt_static_route

This resource provides a means to configure static routes in NSX to determine where IP traffic is routed.

Example Usage

```
resource "nsxt_static_route" "static_route" {
  description       = "SR provisioned by Terraform"
  display_name      = "SR"
  logical_router_id = "${nsxt_logical_tier1_router.router1.id}"
  network           = "4.4.4.0/24"

  next_hop {
    ip_address              = "8.0.0.10"
    administrative_distance = "1"
    logical_router_port_id  = "${nsxt_logical_router_downlink_port.downlink_port.id}"
  }

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `description` - (Optional) Description of this resource.
- `display_name` - (Optional) The display name of this resource. Defaults to ID if not set.
- `tag` - (Optional) A list of scope + tag pairs to associate with this static route.
- `logical_router_id` - (Optional) Logical router id.
- `network` - (Required) CIDR.
- `next_hop` - (Required) List of Next Hops, each with those arguments:
  - `administrative_distance` - (Optional) Administrative Distance for the next hop IP.
  - `ip_address` - (Optional) Next Hop IP.
  - `logical_router_port_id` - (Optional) Reference of logical router port to be used for next hop.

Attributes Reference

In addition to arguments listed above, the following attributes are exported:

- `id` - ID of the static_route.
- `revision` - Indicates current revision number of the object as seen by NSX-T API server. This attribute can be useful for

debugging.

- `next_hop` additional arguments:
  - `bfd_enabled` - Status of bfd for this next hop where bfd_enabled = true indicate bfd is enabled for this next hop and bfd_enabled = false indicate bfd peer is disabled or not configured for this next hop.
  - `blackhole_action` - Action to be taken on matching packets for NULL routes.

Importing

An existing static route can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_static_route.static_route logical-router-uuid/static-route-num
```

The above command imports the static route named `static_route` with the number `static-route-num` that belongs to the tier 1 logical router with the NSX id `logical-router-uuid`.

nsxt_vm_tags

This resource provides a means to configure tags that are applied to objects such as virtual machines. A virtual machine is not directly managed by NSX however, NSX allows attachment of tags to a virtual machine. This tagging enables tag based grouping of objects. Deletion of `nsxt_vm_tags` resource will remove all tags from the virtual machine and is equivalent to update operation with empty tag set.

Example Usage

```
resource "nsxt_vm_tags" "vm1_tags" {
  instance_id = "${vsphere_virtual_machine.vm1.id}"

  tag {
    scope = "color"
    tag   = "blue"
  }
}
```

Argument Reference

The following arguments are supported:

- `instance_id` - (Required) BIOS Id of the Virtual Machine.
- `tag` - (Required) A list of scope + tag pairs to associate with this VM.

Importing

An existing Tags collection can be imported (/docs/import/index.html) into this resource, via the following command:

```
terraform import nsxt_vm_tags.vm1_tags id
```

The above would import NSX virtual machine tags as a resource named `vm1_tags` with the NSX id `id`, where id is external ID (not the BIOS id) of the virtual machine.