Netris docs Documentation

Release Netris v3.0

August 04, 2022

Table of Contents

Netris is the Automatic NetOps platform that runs the physical network and provides cloud-like user experience for NetOps and DevOps engineers.

Netris Tutorials

1.1 Installing a Netris Controller

You can install the Netris controller almost on any 64-bit Linux host. Netris Controller may or may not be on the same network as the managed network nodes are. In fact if there are multiple Netris managed deployments there's no need for an individual controller for each deployment.

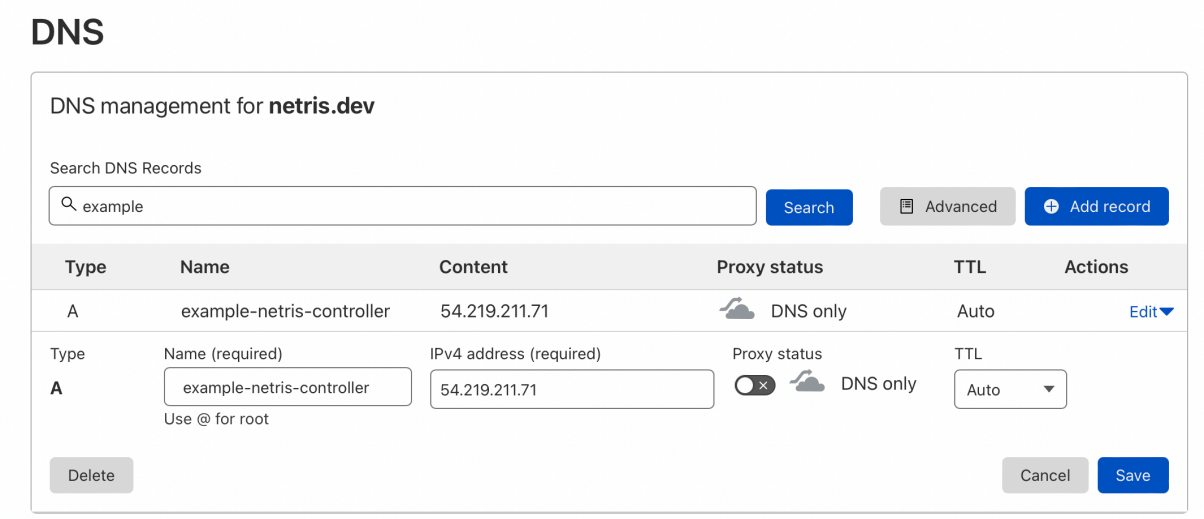
It doesn't matter where to host the Netris controller. What matters is that the Netris controller needs to be accessible over the Internet. So you can access the console, and nodes that are going to be managed by Netris need to have access to the Netris controller through their management network interface.

Linux Host requirements

- RAM: 8 GB
- CPU: 4 Cores
- Disk: 50GB
- OS: Linux 64-bit

In this example I am running my Netris controller on an AWS hosted virtual machine (EC2) which has got a public IP address 54.219.211.71. While it is OK for users and nodes to refer to the Netris Controller through an IP address, I like using a DNS record (this way it will be easier to potentially move Netris Controller somewhere with a different IP address).

I'm using Cloudflare to create this "example-netris-controller.netris.dev" DNS record to point to the public IP address of my EC2 : 54.219.211.71.



Ensure that newly created domain name indeed resolves into the right IP address of the machine that you are going to install the Netris Controller.

To install Netris Controller on a freshly installed Linux you only need to run below one-liner command. Netris Controller installer will stand up a K3S cluster and then will deploy Netris Controller on top of it using Helm Chart. The "–ctl-ssl-issuer" will instruct the installer to generate a Let's Encrypt SSL certificate and the "–ctl-hostname" will hint for what domain name the certificate must be generated. That's why it is important to create the DNS record before this step. For more details, get familiar with the controller installation doc.

```
curl -sfL https://get.netris.ai | sh -s -- --ctl-hostname netris.example.com
--ctl-ssl-issuer letsencrypt
```

Once installation process is finished you will be able to access your newly installed Netris Controller web console using netris/newNet0ps credentials.

Please immediately change the default password to something strong in Setting → My Account → Change Password. You can also use Settings → Login whitelist to restrict web console access to the controller.

2.1 Equinix Metal API integration enablement

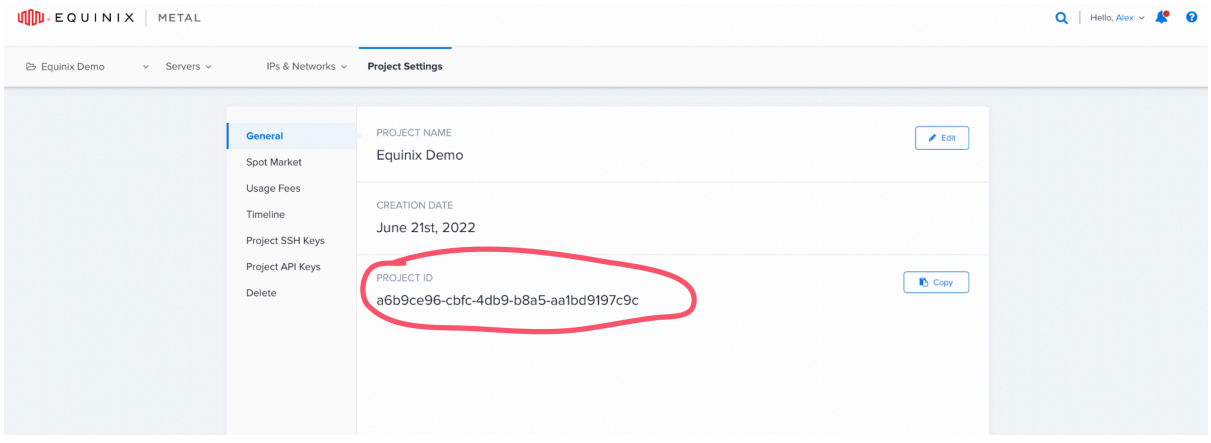
For each Equinix Metal Project+location you need to define an individual Site in Netris Controller.

Go to Netris Web Console → Net → Sites and click +Add.

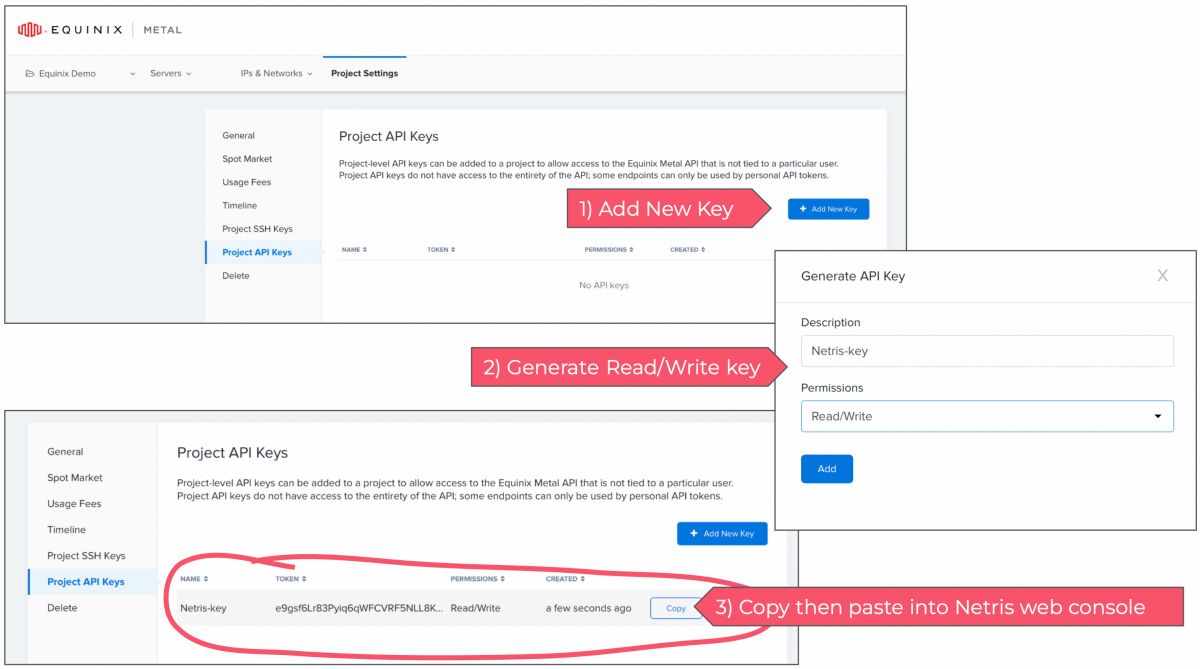
You only need to deal with the below 5 fields. Leave the rest to default values for now.

Netris Parameter	What to do:
Switch Fabric	Select "Equinix Metal" from the dropdown menu.
Name	Type a descriptive name for your Equinix Metal Project+location.
Equinix Project ID	Copy/Paste the Project ID from Equinix Metal portal under Project Settings → General → Project ID.
Equinix Project API key	Create a new Read/Write API key in Equinix Metal portal under Project Settings → Project API keys → + Add New Key. Then copy/paste here.
Equinix Location	Select your equinix location from the dropdown menu.

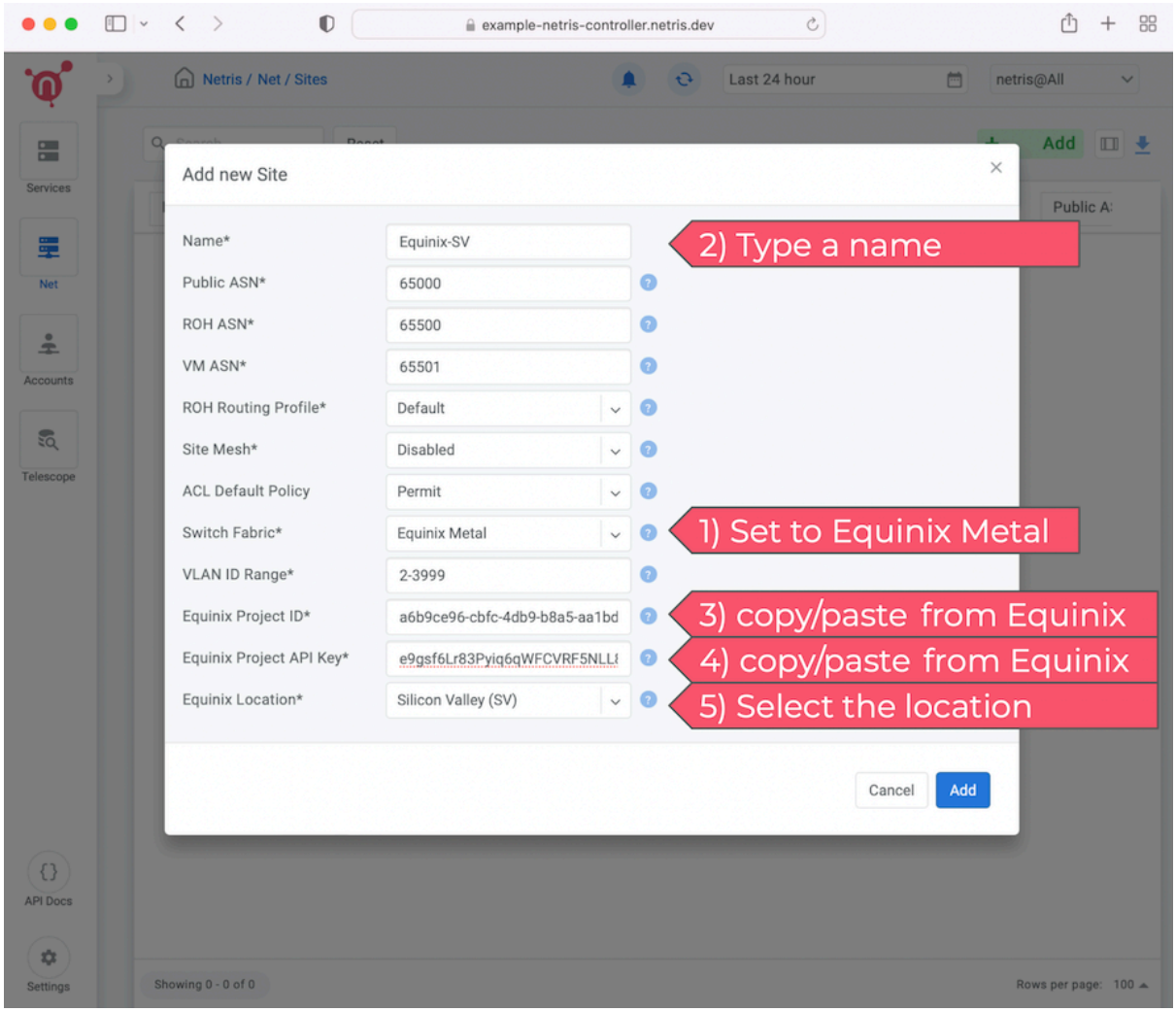
Equinix Metal Project ID



Equinix Metal Project API key



Netris Create New Site

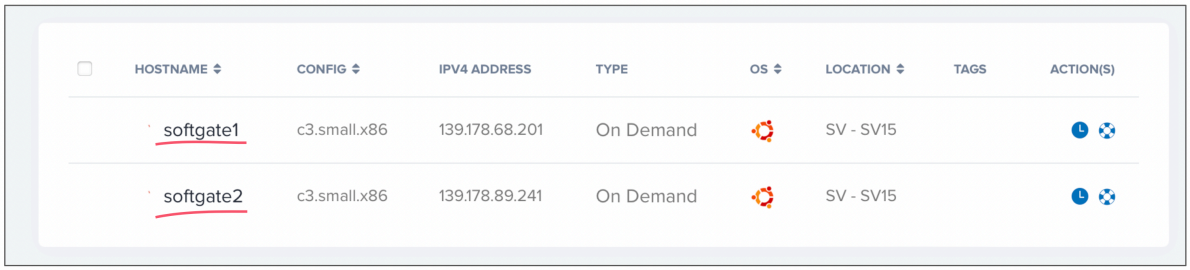


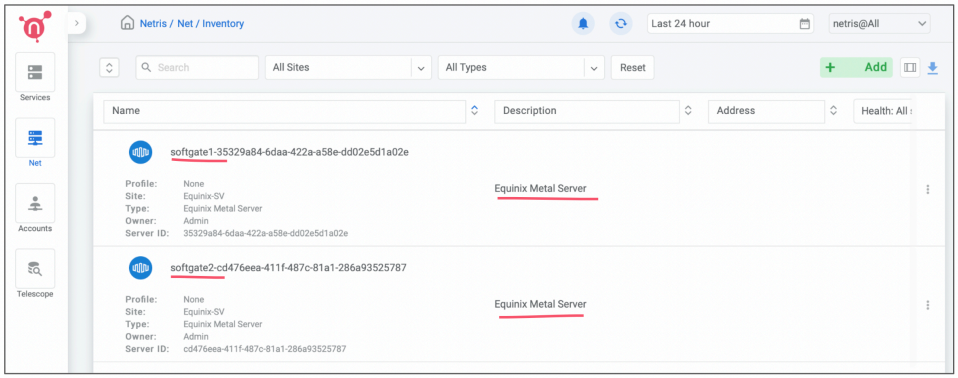
2.2 Provisioning Netris SoftGate nodes in Equinix Metal Project

For SoftGate nodes you can start with two servers of the smallest flavor. In the future if you happen to need to upgrade to high-performance SoftGate PRO (with DPDK acceleration) you can upgrade the servers one-by-one.

Request two servers from Equinix Metal with Ubuntu 22.04 OS and wait until provisioned.

1. At this point you should see Netris Controller listing newly created servers as "Equinix Metal Server" under Netris Web Console → Net → Inventory

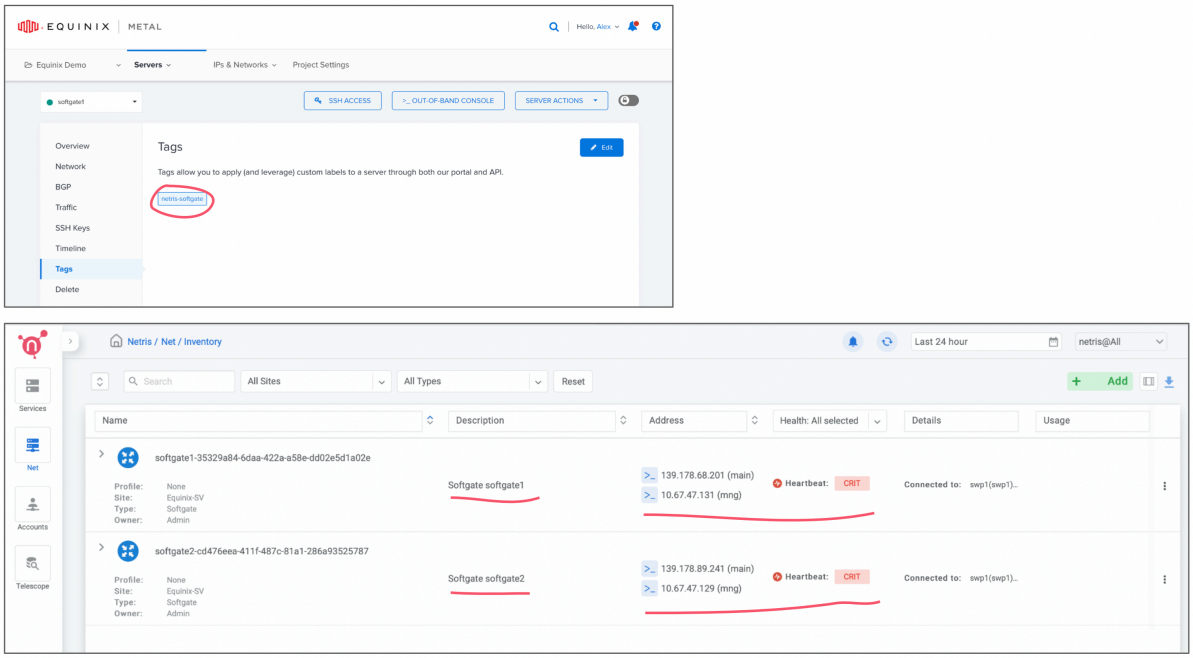




2. When Equinix finishes provisioning of the servers, click on each server name, then click tag, and add a tag "netris-softgate".

   Tag "netris-softgate" will signal Netris Controller that these two servers are going to be used as Netris SoftGate nodes in this particular site (Project+Location).

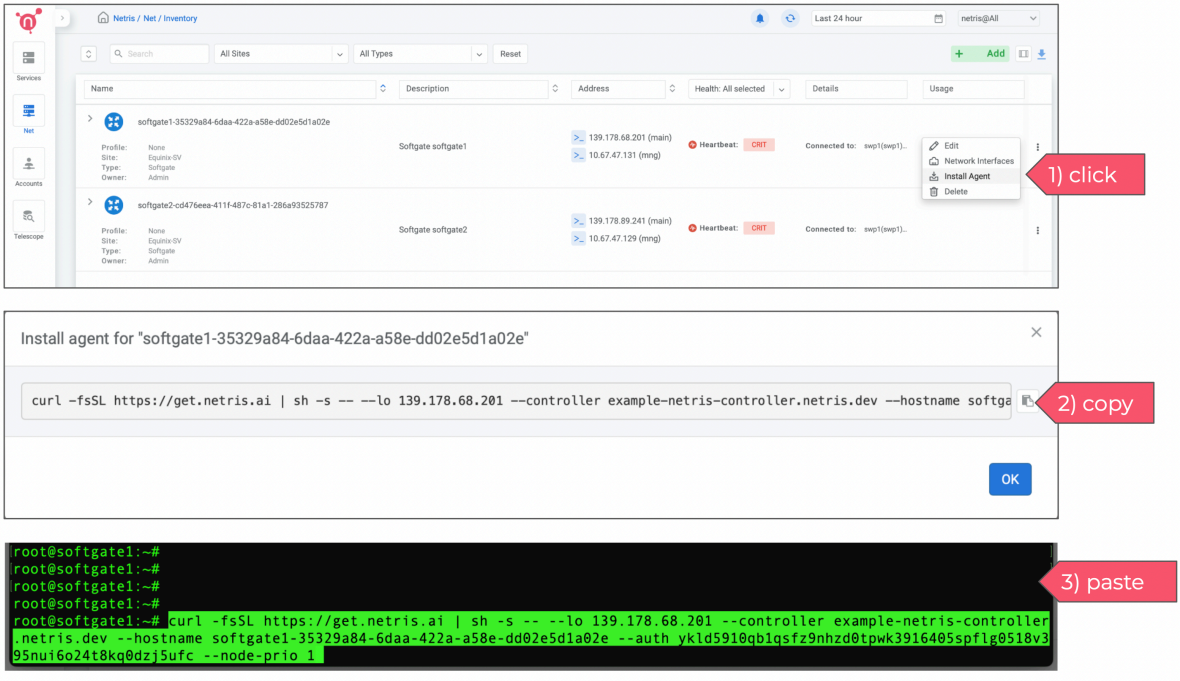
Then you should see in Netris web console that description changes from "Equinix Metal Server" into "Softgate Softgate1(2)". You will also notice IP addresses listed per SoftGate, and Heartbeat: CRIT. We will bring Heartbeat to OK in next step.



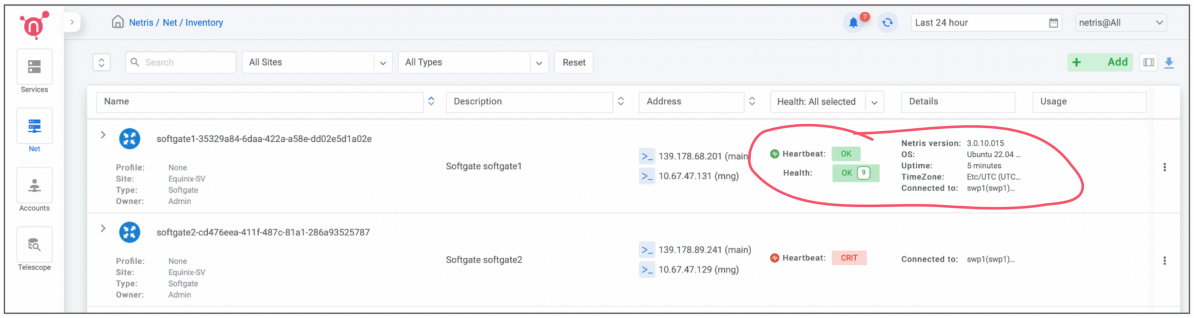
3. Provision SoftGate nodes.

Netris Controller provides a one-liner command for provisioning SoftGate nodes. Go to Netris web console → Net → Inventory, click on the 3 dots menu, and click "Install Agent", and copy the one-liner command.

Then SSH to the corresponding SoftGate server as a root user and paste the one-liner there.



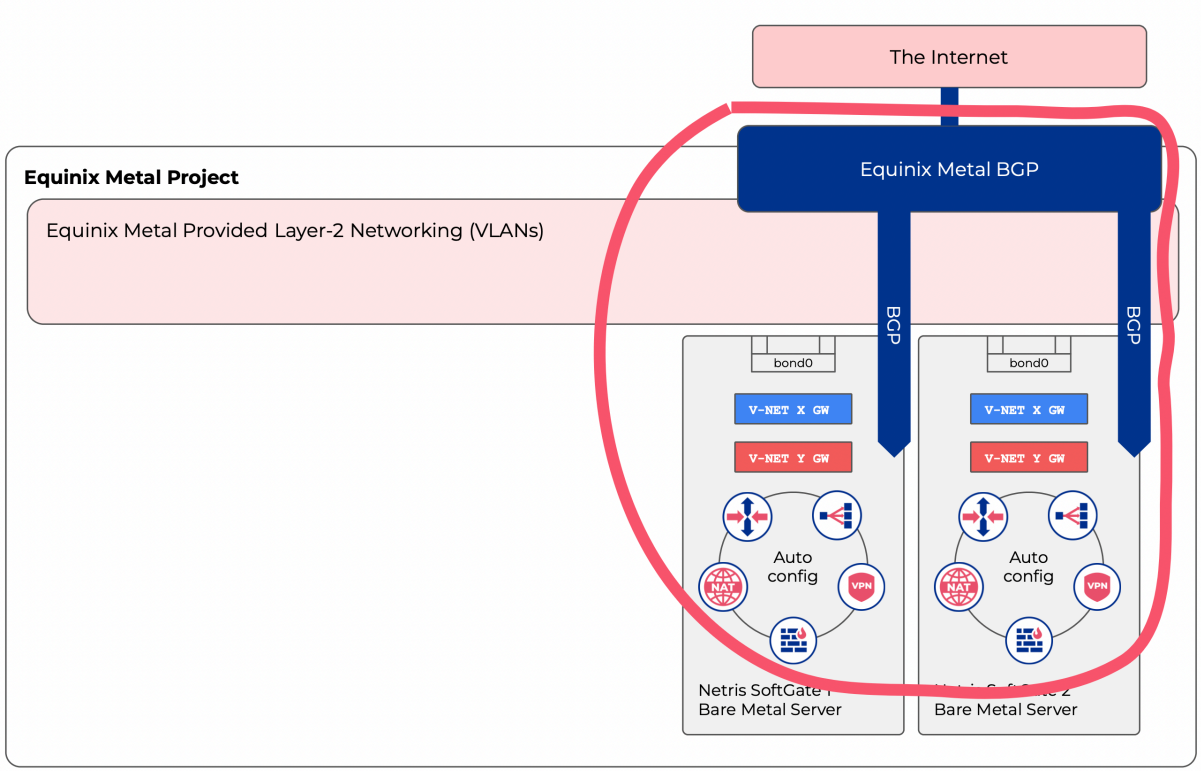
When provisioning is done, reboot the server. In a few minutes Netris Controller should sense the heart beats from SoftGate as in the below screenshot. Repeat for the second SoftGate too.



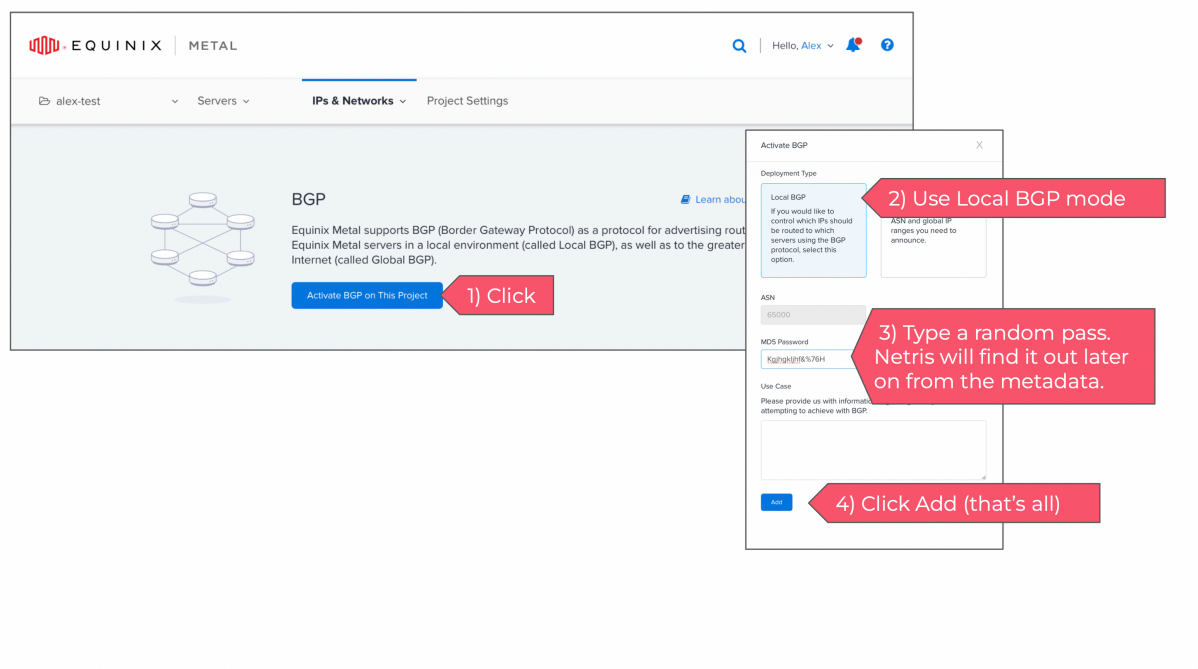
2.3 Activating BGP on Equinix Metal Project

Why use BGP with Equinix Metal? SoftGate nodes are like border routers to your VPC, they are routing traffic between hosts inside your project and the Internet. We are going to establish 2 BGP sessions between SoftGate nodes and Equinix Metal. So there will be 4 BGP sessions total.

We need these BGP sessions for moving further. In the next chapters we are going to request pools of public IP addresses, that Netris will automatically advertise to Equinix Metal, so inbound traffic "knows" how to reach Load Balancer, NAT, and other services that you will use within your VPC.

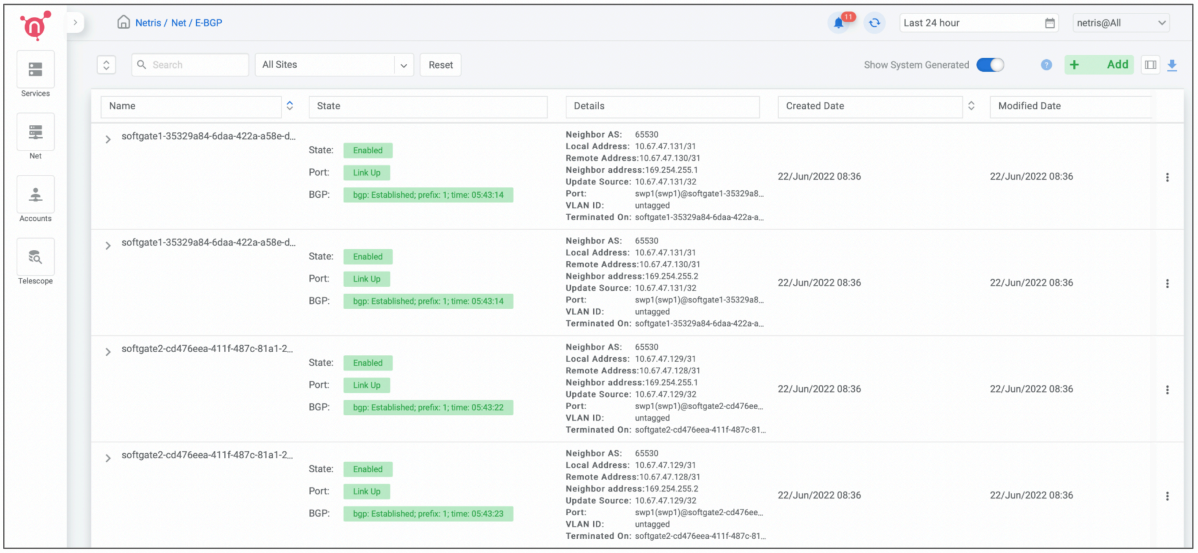


You only need to activate BGP on the Equinix Metal Project. Netris will handle the rest. In the Equinix Metal web console go to IPs & Networks → BGP then click Activate BGP on This Project. (see below screenshots)



Netris will handle the rest behind the scenes automatically. Netris will enable BGP peering on the Equinix Metal side, Netris will pull the metadata with the BGP info, and will automatically configure FRR (Free Range Routing BGP daemon) on both SoftGate nodes to bring up the BGP sessions up.

After a few minutes you should see 4 new BGP sessions in your Netris web console under Net → E-BGP. (example screenshot below).



Now your Netris VPC has established BGP sessions with Equinix Metal Project, and you can proceed to the next step.

2.4 Enabling services (NAT, V-Net, Load Balancer, IP pools)

Although bare metal servers in Equinix Metal Project get a public IP address and can access the Internet, it's not the case with the VMs. When you use server virtualization something needs to be the gateway for your virtual networks. That gateway will provide packet forwarding with access

control between your virtual private networks, will provide NAT, and on-demand (Elastic) Load Balancer services. Physically that gateway is Netris SoftGate. And it operates automatically, providing you with the VPC-like networking capabilities.

Both NAT and on-demand Load Balancer services need public IP addresses.

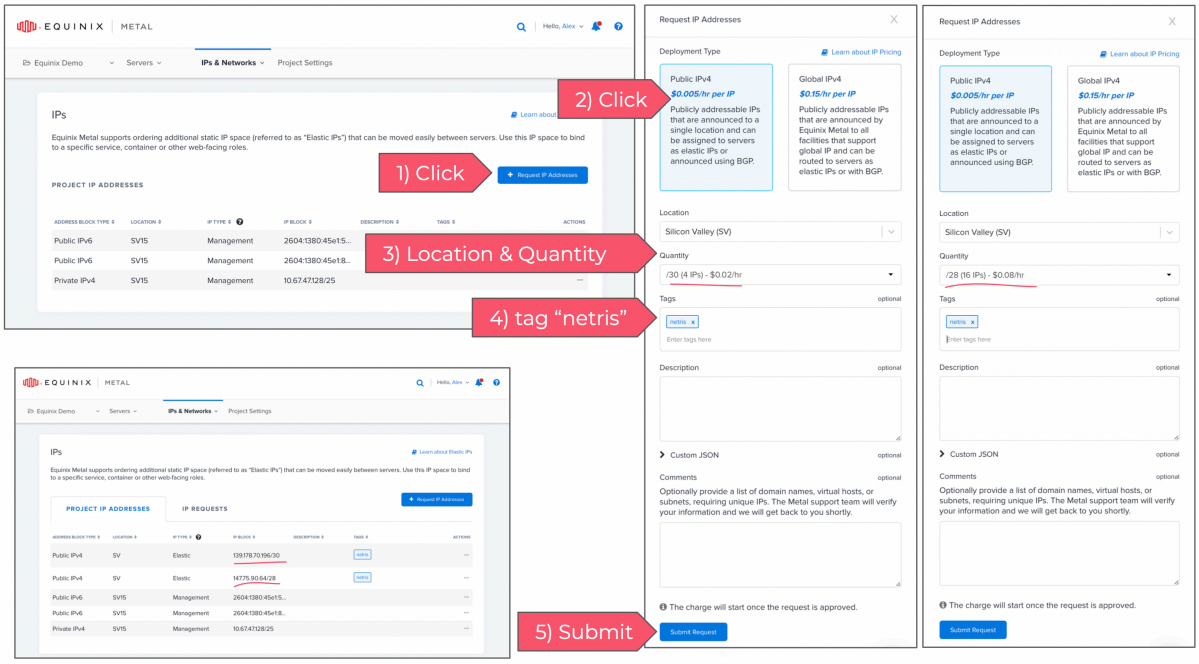
2.4.1 1) Requesting new Public IP address block

Go to Equinix Metal web console and click on IPs & Networks → IPs (see the screenshot below)

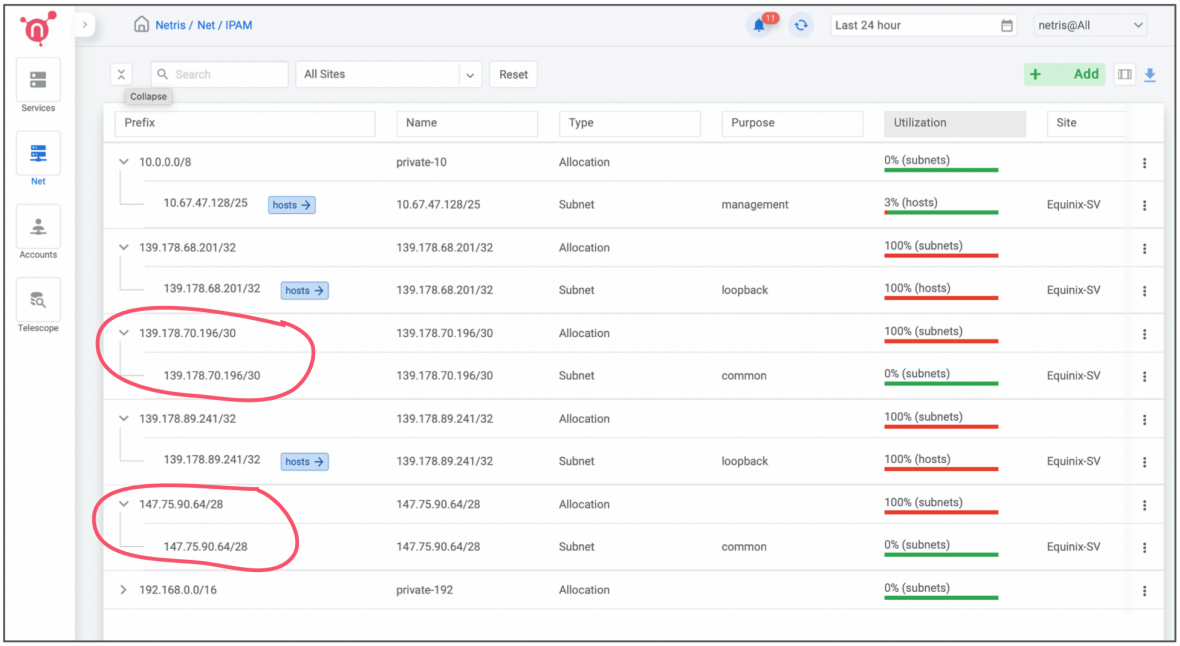
In this example, I'm requesting two IP address blocks, one /30 (4 IPs) for NAT and one /28 (16 IPs) for Load Balancer.

It's important to tag IP blocks as "netris". This is a signal for Netris Controller that this block is intended for Netris VPC.

You can always request more IP address blocks in the future. Also it is possible to request a large block and then use Netris IPAM for crushing it into smaller blocks. You can read more about Netris IPAM in Netris docs.



Once IP address blocks are provisioned on Equinix Metal Project you should be able to find them automatically replicated in Netris web console under Net → IPAM

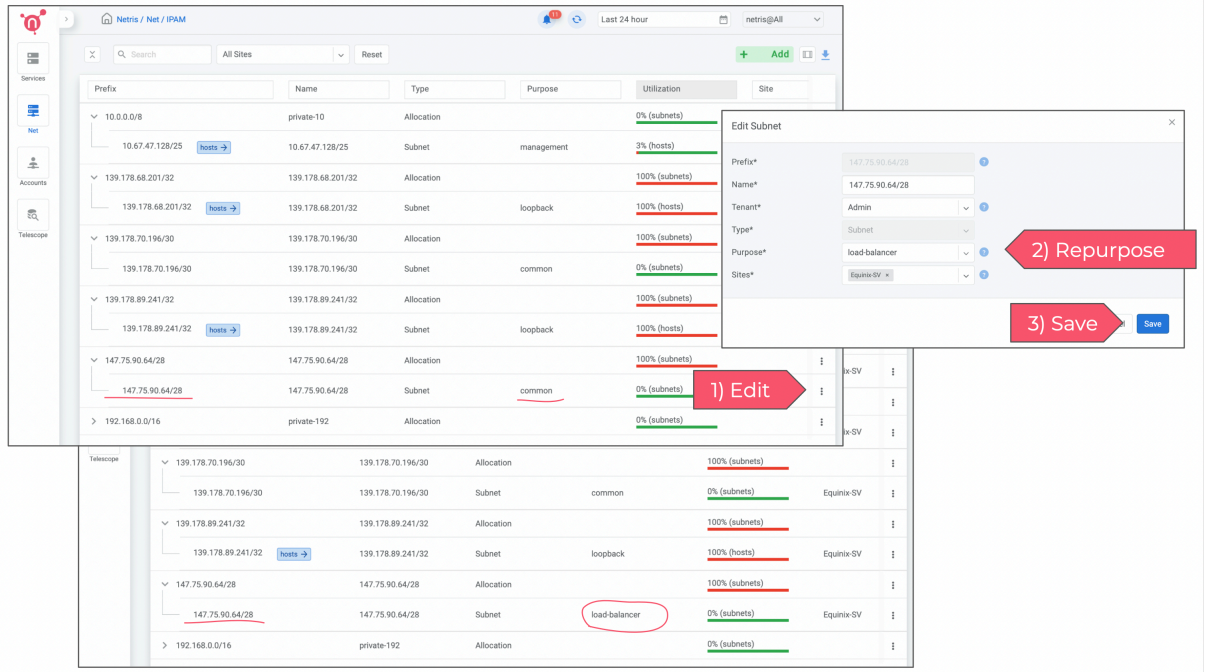


You don't need to worry about advertising them over BGP, Netris will handle that automatically when that makes sense (associated with any service).

2.4.2 2) Enable on-demand (elastic) Load Balancer

To Enable on-demand (elastic) Load Balancer you only need to change the "purpose" field of appropriate IP address block from "common" into "load-balancer"

Click on the 3 dots menu (in this example of /28 IP address block), click edit, and select "load-balancer" from the dropdown menu next to the "purpose" field.



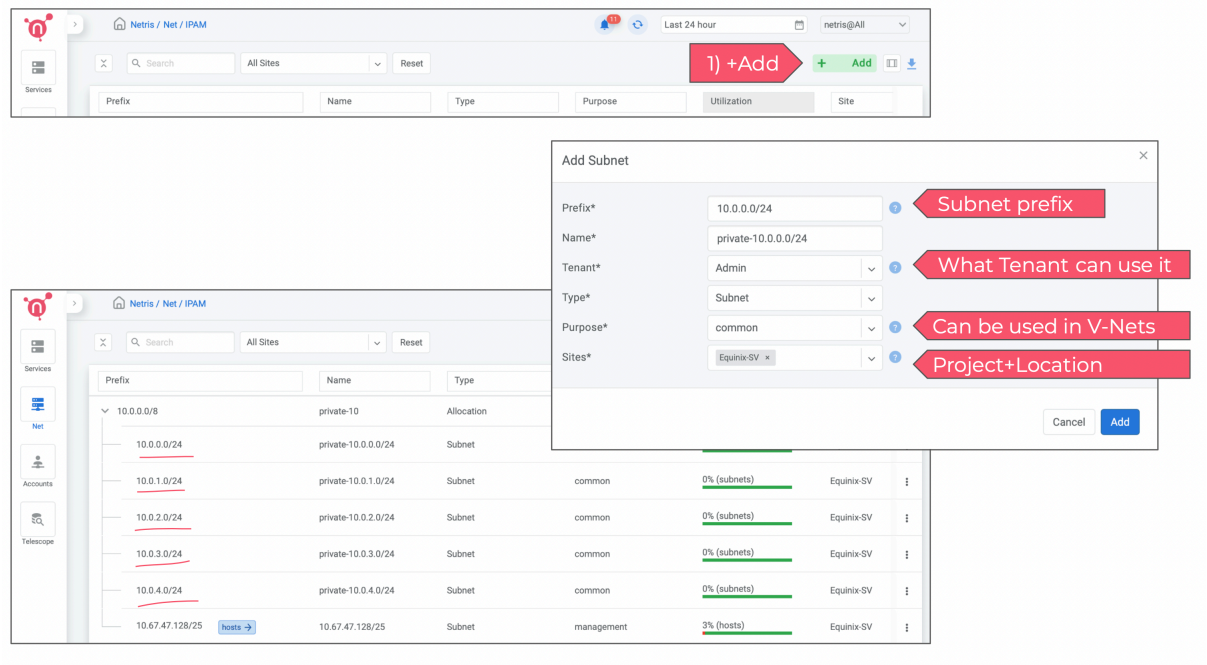
Now on-demand (elastic) load balancer service is enabled and can be consumed either from web console, or from Kubernetes using service of the type load-balancer, or with Terraform.

Please note that in this example we left the field Tenant set to Admin. Tenancy is used for role based

access control and resource delegation. In other words you may want to create a user role and tenant for your colleagues that are supposed to consume Netris VPC services, but not administer it.

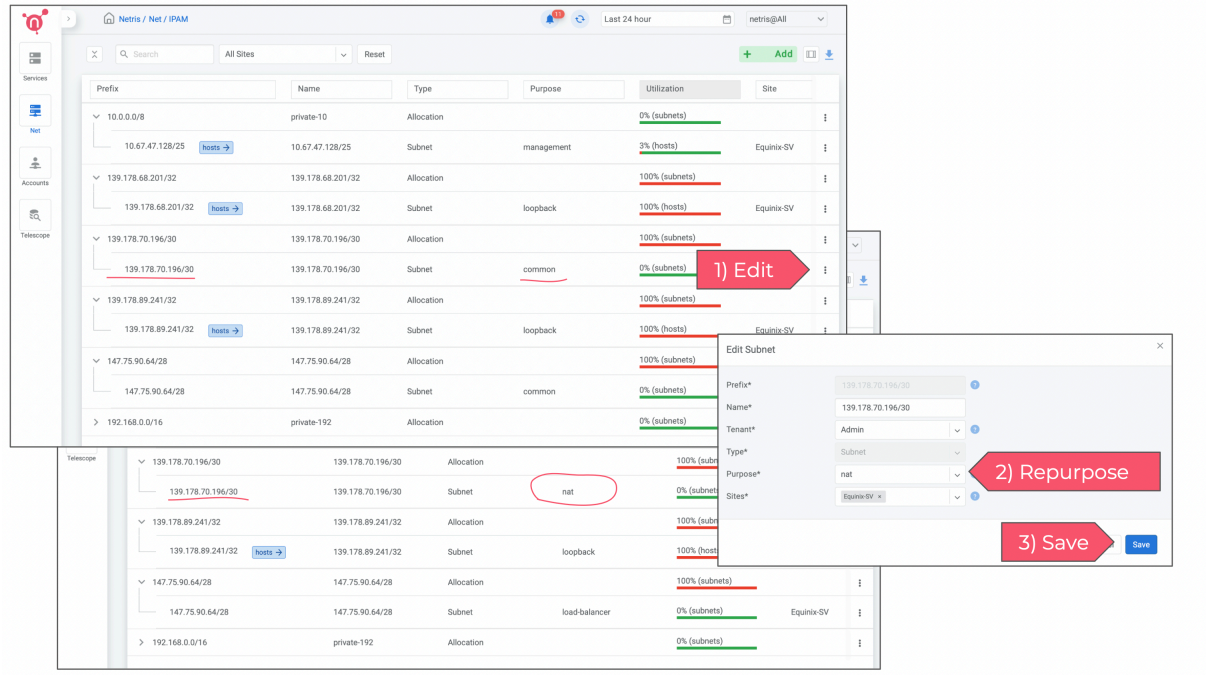
2.4.3 3) Enable V-Net

V-Net is a service for virtual private networks. You need to create a few subnets of private IP address blocks to be used by you and your colleagues later on for creating V-Nets. For now you can set the Tenant field to Admin, but in the future if you need to create a separate user role that should be able to consume Netris VPC but not administer it, you will need to create a separate Tenant and give that Tenant IP resources.

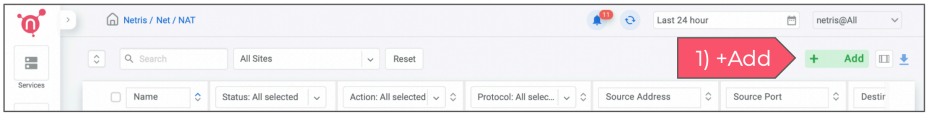


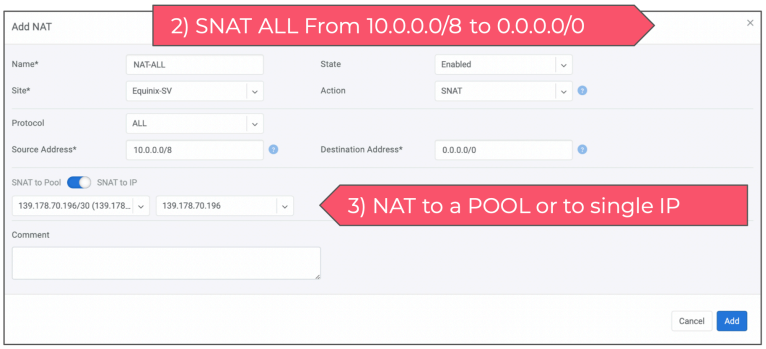
2.4.4 4) Enable NAT

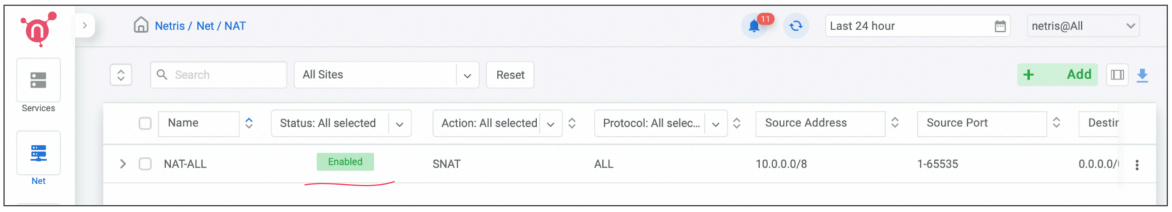
To enable NAT, you need to repurpose a block of IP addresses for NAT. In the below example I'm repurposing the newly requested /30 subnet for NAT.



Then You need to create a NAT rule. In the Net → NAT section of Netris web console. Netris supports most of the standard rules for SNAT and DNAT. In this example I'm enabling SNAT for the entire 10.0.0.0/8 private network, so basically I just want to ensure that VMs that will get IPs from private networks will get outbound Internet access through NAT. You can always have more granular control either through NAT rule or using Services → ACLs.







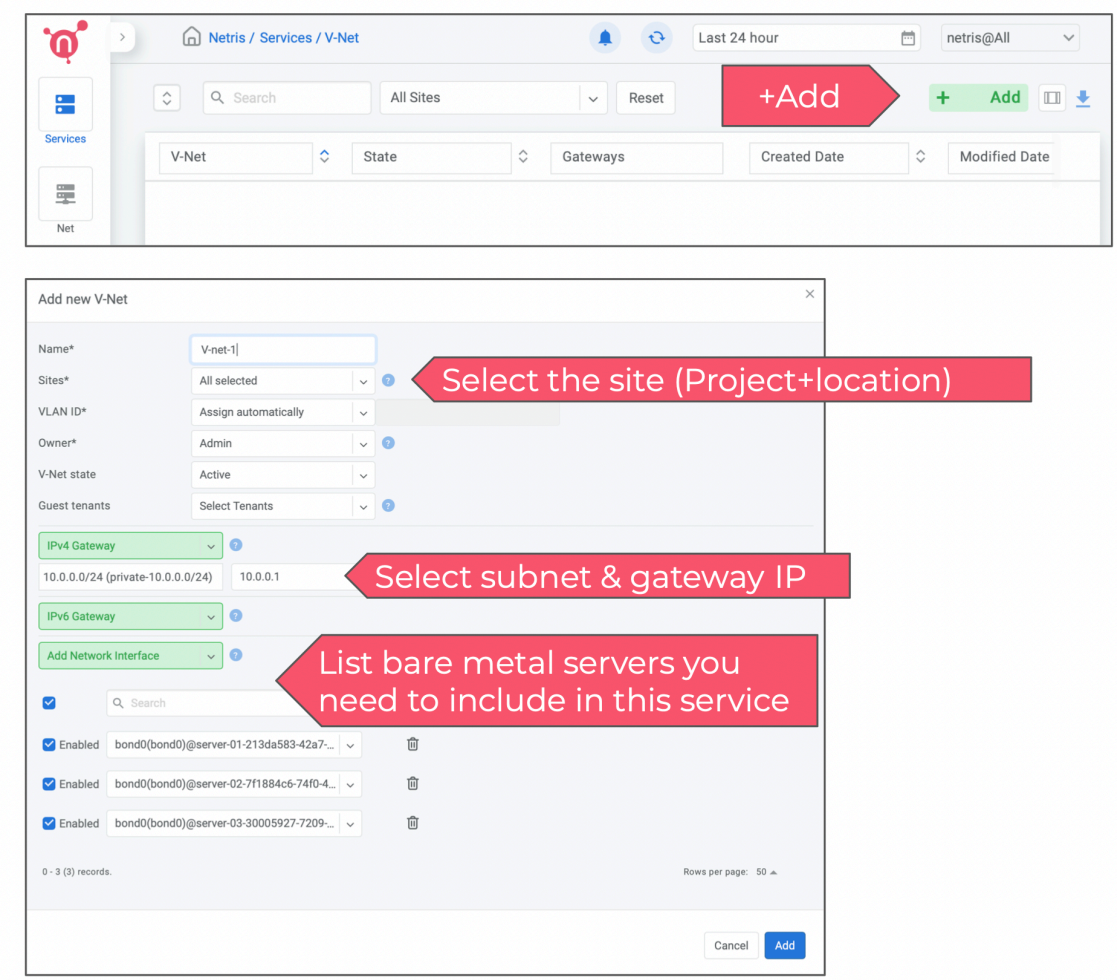
At this point the minimal configuration of Netris VPC networking is done, next chapters will describe how to consume the VPC, how to request resources and services.

2.5 Using V-Net (isolated virtual network) services in Equinix Metal Project

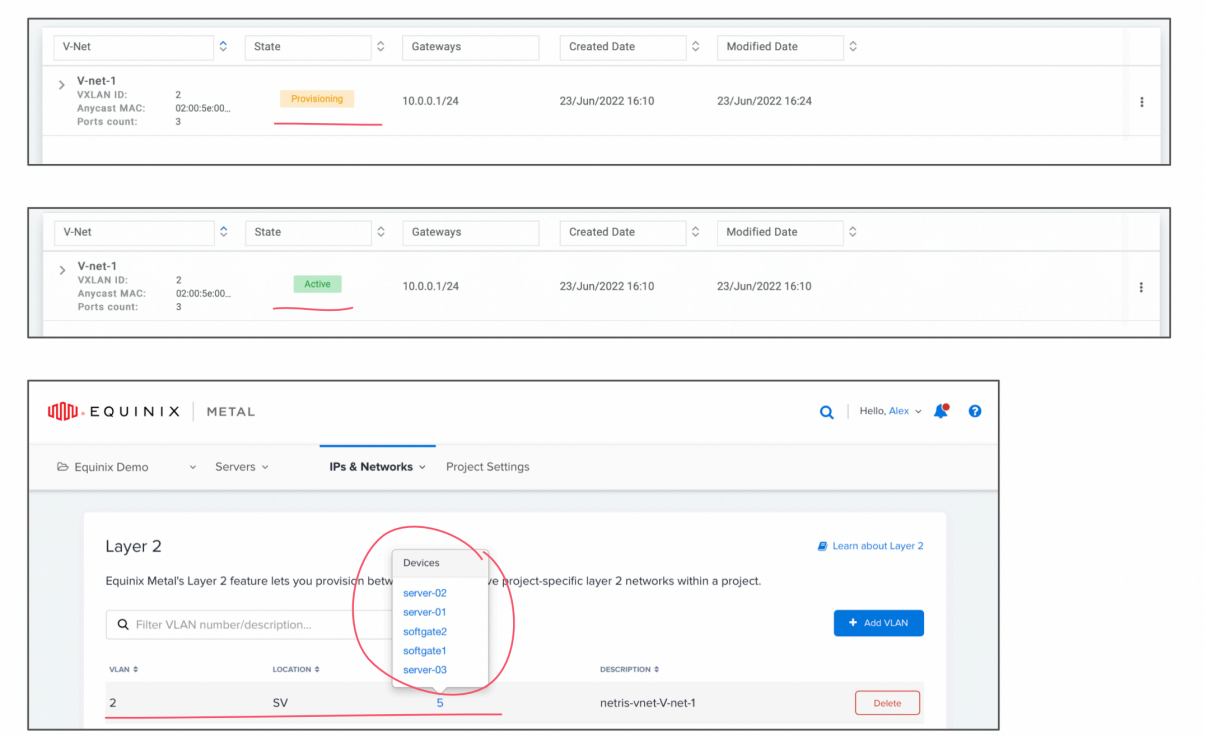
V-Net, under Services –> V-Net, is an isolated virtual network service. Netris loads your bare metal server metadata from Equinix Metal Project into the Netris database. So when you create a V-Net service (a virtual network), you list the bare-metal servers you need to get on that virtual network.

Netris V-Net (in Equinix Metal scenario) has a global VLAN ID. Per every V-Net, Netris will provision a Layer-2 network using Equinix Metal API. Then Netris will include the listed bare metal servers + SoftGate nodes into that newly created Layer-2 network. SoftGate nodes are the default gateway for the V-Net services.

You should create a corresponding V-Net for each virtual network if you use Vmware, KVM, or any other server virtualization platform or VLANs in any way. VLAN ID will be the unique identifier between Netris, Equinix, and your Compute.



In this example, the new V-NET has VLAN ID 2, subnet 10.0.0.0/24, and gateway 10.0.0.1. That means three servers (server-01, server-02, server-03) can launch VMs (or subinterfaces) into a virtual network with VLAN ID 2, and they should use IP addresses from 10.0.0.2-254 pointing to 10.0.0.1 as the default gateway. Netris SoftGate will serve that traffic, and since we have enabled NAT globally in previous chapters, hosts living in VLAN 2 will have Internet access over the NAT.



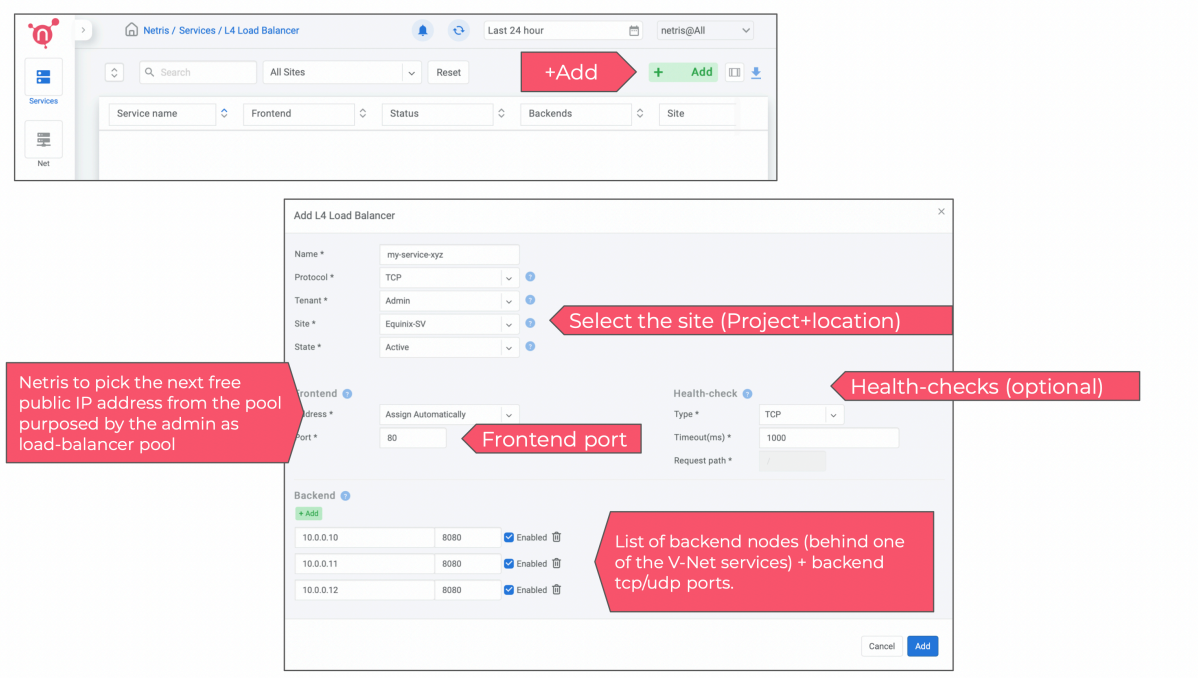
Note that you can use Services –> ACLs for granular control over traffic between multiple V-NETs as well as to/from outside (Internet or other. remote sites)

2.6 Using on-demand (elastic) L4 Load Balancer service

Services –> L4 Load Balancer is an on-demand (elastic) server load balancer. You can natively use it for Kubernetes, as well as for any TCP/UDP service.

Please check our cloud-native tools section of the documentation portal for consuming Load Balancer using Kubernetes native method, Kubernetes CRDs, or Terraform.

Meanwhile, below is a screenshot of simply requesting a Load Balancer service through the Netris web console.


Netris / Services / L4 Load Balancer
Last 24 hour
netris@All
Search
All Sites
Reset
+Add
Add
Service name
Frontend
Status
Backends
Site
Services
Net
Add L4 Load Balancer
Name *
my-service-xyz
Protocol *
TCP
Tenant *
Admin
Site *
Equinix-SV
State *
Active
Select the site (Project+location)
Netris to pick the next free public IP address from the pool purposed by the admin as load-balancer pool
Health-checks (optional)
Assign Automatically
80
Frontend port
Health-check
Type *
TCP
Timeout(ms) *
1000
Request path *
Backend
+ Add
10.0.0.10
8080
Enabled
10.0.0.11
8080
Enabled
10.0.0.12
8080
Enabled
List of backend nodes (behind one of the V-Net services) + backend tcp/udp ports.
Cancel
Add

Chapter 3

Kubernetes Integration

Netris integrates with Kube API to provide on-demand load balancer and other Kubernetes specific networking features. Netris-Kubernetes integration is designed to complement Kubernetes CNI networking and provide a cloud-like user experience to local Kubernetes clusters.

3.1 Install Netris Operator

Integration between the Netris Controller and the Kubernetes API is completed by installing the Netris Operator. Installation can be accomplished by installing regular manifests or a helm chart:

3.1.1 Helm Chart Method

Instructions are available on Github: https://github.com/netrisai/netris-operator/tree/master/deploy/charts/netris-operator#installing-the-chart

3.1.2 Regular Manifest Method

1. Install the latest Netris Operator:

```
kubectl apply -f
https://github.com/netrisai/netris-operator/releases/latest/download/netris-operator.yaml
```

2. Create credentials secret for Netris Operator:

```
kubectl -nnetris-operator create secret generic netris-creds \
--from-literal=host='http://**your-netris-controller-ip-or-host**' \
--from-literal=login='**your-netris-admin-username**'
--from-literal=password='**your-netris-admin-password**'
```

3. Inspect the pod logs and make sure the operator is connected to Netris Controller:

```
kubectl -nnetris-operator logs -l netris-operator=controller-manager
--all-containers -f
```

Example output demonstrating the successful operation of Netris Operator:

```
{"level":"info","ts":1629994653.6441543,"logger":"controller","msg":"Starting
workers","reconcilerGroup":"k8s.netris.ai","reconcilerKind":"L4LB","controller":"l4lb","worker
count":1}
```

Note After installing the Netris Operator, your Kubernetes cluster and physical network control planes are connected.

3.2 Using Type 'LoadBalancer'

In this scenario we will be installing a simple application that requires a network load balancer:

Install the application "Podinfo":

```
kubectl apply -k github.com/stefanprodan/podinfo/kustomize
```

Get the list of pods and services in the default namespace:

```
kubectl get po,svc
```

As you can see, the service type is "ClusterIP":

```
NAME                              READY   STATUS    RESTARTS   AGE
pod/podinfo-576d5bf6bd-7z9jl      1/1     Running   0          49s
pod/podinfo-576d5bf6bd-nhlmh      1/1     Running   0          33s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/podinfo      ClusterIP   172.21.65.106   <none>        9898/TCP,9999/TCP
50s
```

In order to request access from outside, change the type to "LoadBalancer":

```
kubectl patch svc podinfo -p '{"spec":{"type":"LoadBalancer"}}'
```

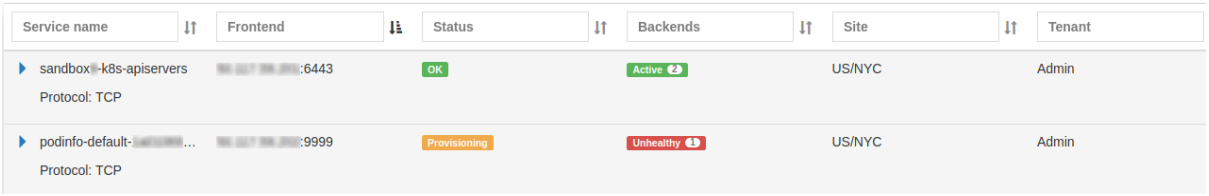
Check the services again:

```
kubectl get svc
```

Now we can see that the service type changed to LoadBalancer, and "EXTERNAL-IP" switched to pending state:

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
podinfo      LoadBalancer   172.21.65.106   <pending>
9898:32584/TCP,9999:30365/TCP   8m57s
```

Going into the Netris Controller web interface, navigate to **Services / L4 Load Balancer**, and you may see L4LBs provisioning in real-time. If you do not see the provisioning process it is likely because it already completed. Look for the service with the name **"podinfo-xxxxxxxx"**



After provisioning has finished, inspect the service in k8s:

```
kubectl get svc
```

You can see that "EXTERNAL-IP" has been injected into Kubernetes:

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
podinfo      LoadBalancer   172.21.65.106   50.117.59.202
9898:32584/TCP,9999:30365/TCP   9m17s
```

3.3 Using Netris Custom Resources

3.3.1 Introduction to Netris Custom Resources

In addition to provisioning on-demand network load balancers, Netris Operator can also provide automatic creation of network services based on Kubernetes CRD objects. Let's take a look at a few common examples:

3.3.2 L4LB Custom Resource

In the previous section, when we changed the service type from "ClusterIP" to "LoadBalancer", Netris Operator detected a new request for a network load balancer, then it created L4LB custom resources. Let's see them:

```
kubectl get l4lb
```

As you can see, there are two L4LB resources, one for each podinfo's service port:

```
NAME                                                                STATE     FRONTEND
PORT        SITE      TENANT    STATUS    AGE
podinfo-default-66d44feb-0278-412a-a32d-73afe011f2c6-tcp-9898   active
50.117.59.202   9898/TCP    US/NYC    Admin     OK        33m
podinfo-default-66d44feb-0278-412a-a32d-73afe011f2c6-tcp-9999   active
50.117.59.202   9999/TCP    US/NYC    Admin     OK        32m
```

You can't edit/delete them, because Netris Operator will recreate them based on what was originally deployed in the service specifications.

Instead, let's create a new load balancer using the CRD method. This method allows us to create L4 load balancers for services outside of what is being created natively with the Kubernetes service schema. Our new L4LB's backends will be "srv04-nyc" & "srv05-nyc" on TCP port 80. These servers are already running the Nginx web server, with the hostname present in the index.html file.

Create a yaml file:

```
cat << EOF > srv04-5-nyc-http.yaml
apiVersion: k8s.netris.ai/v1alpha1
kind: L4LB
metadata:
 name: srv04-5-nyc-http
spec:
 ownerTenant: Admin
 site: US/NYC
 state: active
 protocol: tcp
 frontend:
   port: 80
 backend:
   - 192.168.45.64:80
   - 192.168.46.65:80
 check:
   type: tcp
   timeout: 3000
EOF
```

And apply it:

```
kubectl apply -f srv04-5-nyc-http.yaml
```

Inspect the new L4LB resources via kubectl:

```
kubectl get l4lb
```

As you can see, provisioning started:

```
NAME                                                                STATE     FRONTEND
PORT        SITE      TENANT    STATUS          AGE
podinfo-default-d07acd0f-51ea-429a-89dd-8e4c1d6d0a86-tcp-9898    active
50.117.59.202    9898/TCP    US/NYC    Admin     OK              2m17s
podinfo-default-d07acd0f-51ea-429a-89dd-8e4c1d6d0a86-tcp-9999    active
50.117.59.202    9999/TCP    US/NYC    Admin     OK              3m47s
srv04-5-nyc-http                                                    active
50.117.59.203    80/TCP      US/NYC    Admin     Provisioning    6s
```

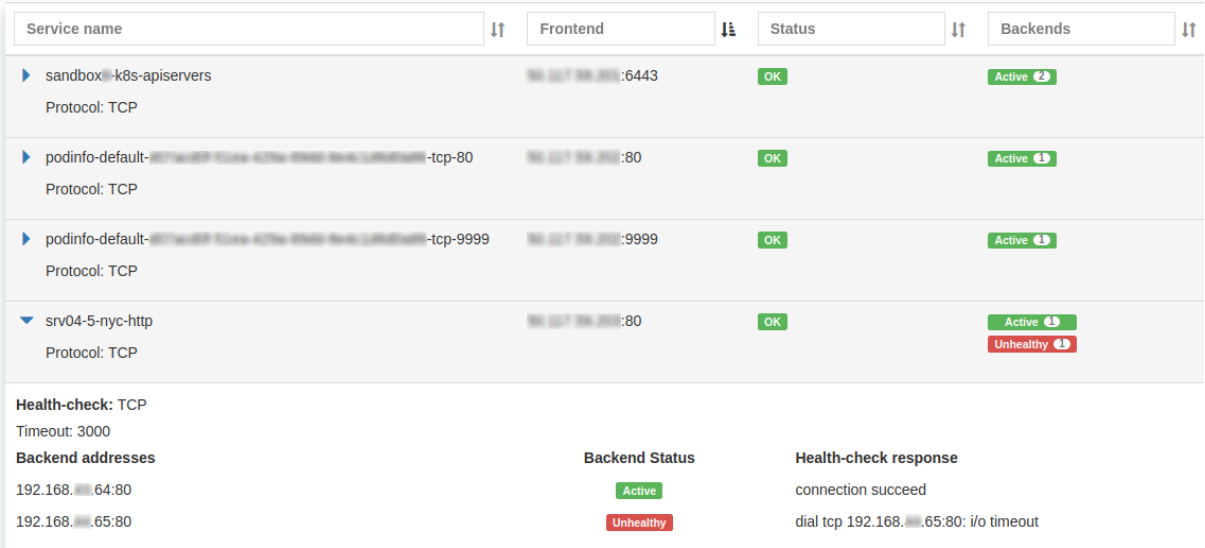
When provisioning is finished, you should be able to connect to L4LB. Try to curl, using the L4LB frontend address displayed in the above command output:

```
curl 50.117.59.203
```

You will see the servers' hostname in curl output:

```
SRV04-NYC
```

You can also inspect the L4LB in the Netris Controller web interface:



3.3.3 V-Net Custom Resource

You can also create Netris V-Nets (L2 segments) via Kubernetes with a simple manifest:

```
cat << EOF > vnet-customer.yaml
apiVersion: k8s.netris.ai/v1alpha1
kind: VNet
metadata:
 name: vnet-customer
spec:
 ownerTenant: Admin
 guestTenants: []
 sites:
   - name: US/NYC
     gateways:
       - 192.168.46.1/24
     switchPorts:
       - name: swp2@sw22-nyc
EOF
```

And apply it:

```
kubectl apply -f vnet-customer.yaml
```

Let's check our VNet resources in Kubernetes:

```
kubectl get vnet
```

As you can see, provisioning for our new VNet has started:

```
NAME            STATE    GATEWAYS          SITES    OWNER   STATUS          AGE
vnet-customer   active   192.168.46.1/24   US/NYC   Admin   Provisioning    7s
```

After provisioning has completed, the L4LB's checks should work for both backend servers, and incoming requests should be balanced between them.

3.3.4 BGP Custom Resource

You can create BGP peers via Kubernetes manifests:

1. Create a yaml file:

```
cat << EOF > isp2-customer.yaml
apiVersion: k8s.netris.ai/v1alpha1
kind: BGP
metadata:
  name: isp2-customer
spec:
  site: US/NYC
  softgate: SoftGate2
  neighborAs: 65007
  transport:
    name: swp14@sw02-nyc
    vlanId: 1092
  localIP: 50.117.59.118/30
  remoteIP: 50.117.59.117/30
  description: Example BGP to ISP2
  prefixListOutbound:
    - permit 50.117.59.192/28 le 32
EOF
```

2. Apply the manifest file:

```
kubectl apply -f isp2-customer.yaml
```

3. Check created BGP:

```
kubectl get bgp
```

Allow up to 1 minute for both sides of the BGP sessions to come up:

```
NAME            STATE     BGP STATE   PORT STATE   NEIGHBOR AS   LOCAL ADDRESS
REMOTE ADDRESS     AGE
isp2-customer   enabled                            65007         50.117.59.118/30
50.117.59.117/30   15s
```

Then check the state again:

```
kubectl get bgp
```

The output is similar to this:

```
NAME            STATE     BGP STATE                                          PORT
```

```
STATE   NEIGHBOR AS   LOCAL ADDRESS      REMOTE ADDRESS     AGE
isp2-customer   enabled   bgp: Established; prefix: 30; time: 00:00:51   UP
65007         50.117.59.118/30   50.117.59.117/30   2m3s
```

Feel free to use the import annotation for this BGP if you created it from the controller web interface previously.

Return to the Netris UI and navigate to **Net / Topology** to see the new BGP neighbor you created.

3.3.5 Importing existing resources from Netris Controller to Kubernetes

You can import any custom resources, already created from the Netris Controller to k8s by adding this annotation:

```
resource.k8s.netris.ai/import: "true"
```

Otherwise, if try to apply them w/out "import" annotation, the Netris Operator will complain that the resource with such name or specs already exists.

After importing resources to k8s, they will belong to the Netris Operator, and you won't be able to edit/delete them directly from the Netris Controller web interface, because the Netris Operator will put everything back, as declared in the custom resources.

3.3.6 Reclaim Policy

There is also one useful annotation. So suppose you want to remove some custom resource from k8s, and want to prevent its deletion from the controller, for that you can use "reclaimPolicy" annotation:

```
resource.k8s.netris.ai/reclaimPolicy: "retain"
```

Just add this annotation in any custom resource while creating it. Or if the custom resource has already been created, change the `"delete"` value to `"retain"` for key `resource.k8s.netris.ai/reclaimPolicy` in the resource annotation. After that, you'll be able to delete any Netris Custom Resource from Kubernetes, and it won't be deleted from the controller.

See also

See all options and examples for Netris Custom Resources here.

3.4 Calico CNI Integration

Netris Operator can integrate with Calico CNI. This annotation will automatically create BGP peering between cluster nodes and the leaf/TOR switch for each node, then to clean up it will disable Calico Node-to-Node mesh. To understand why you need to configure peering between Kubernetes nodes and the leaf/TOR switch, and why you should disable Node-to-Node mesh, review the calico docs.

Integration is very simple, just need to add the annotation in calico's `bgpconfigurations` custom resource. Before doing that, let's see the current state of `bgpconfigurations`:

```
kubectl get bgpconfigurations default -o yaml
```

As we can see, `nodeToNodeMeshEnabled` is enabled, and `asNumber` is 64512 (it's Calico default AS number):

```
apiVersion: crd.projectcalico.org/v1
kind: BGPConfiguration
metadata:
 annotations:
   ...
 name: default
```

```
  ...
spec:
 asNumber: 64512
 logSeverityScreen: Info
 nodeToNodeMeshEnabled: true
```

Let's enable the "netris-calico" integration:

```
kubectl annotate bgpconfigurations default manage.k8s.netris.ai/calico='true'
```

Let's check our BGP resources in k8s:

```
kubectl get bgp
```

Here are our freshly created BGPs, one for each k8s node:

```
NAME                                  STATE     BGP STATE
PORT STATE   NEIGHBOR AS   LOCAL ADDRESS      REMOTE ADDRESS      AGE
isp2-customer                         enabled   bgp: Established; prefix: 28; time:
00:06:18   UP           65007         50.117.59.118/30   50.117.59.117/30    7m59s
sandbox9-srv06-nyc-192.168.110.66   enabled
4200070000    192.168.110.1/24   192.168.110.66/24   26s
sandbox9-srv07-nyc-192.168.110.67   enabled
4200070001    192.168.110.1/24   192.168.110.67/24   26s
sandbox9-srv08-nyc-192.168.110.68   enabled
4200070002    192.168.110.1/24   192.168.110.68/24   26s
```

You might notice that peering neighbor AS is different from Calico's default 64512. The is because the Netris Operator is setting a particular AS number for each node.

Allow up to 1 minute for the BGP sessions to come up, then check BGP resources again:

```
kubectl get bgp
```

As seen our BGP peers are established:

```
NAME                                  STATE     BGP STATE
PORT STATE   NEIGHBOR AS   LOCAL ADDRESS      REMOTE ADDRESS      AGE
isp2-customer                         enabled   bgp: Established; prefix: 28; time:
00:07:48   UP           65007         50.117.59.118/30   50.117.59.117/30    8m41s
sandbox9-srv06-nyc-192.168.110.66   enabled   bgp: Established; prefix: 5; time:
00:00:44    N/A          4200070000    192.168.110.1/24   192.168.110.66/24   68s
sandbox9-srv07-nyc-192.168.110.67   enabled   bgp: Established; prefix: 5; time:
00:00:19    N/A          4200070001    192.168.110.1/24   192.168.110.67/24   68s
sandbox9-srv08-nyc-192.168.110.68   enabled   bgp: Established; prefix: 5; time:
00:00:44    N/A          4200070002    192.168.110.1/24   192.168.110.68/24   68s
```

Now let's check if `nodeToNodeMeshEnabled` is still enabled:

```
kubectl get bgpconfigurations default -o yaml
```

It is disabled, which means the "netris-calico" integration process is finished:

```
apiVersion: crd.projectcalico.org/v1
kind: BGPConfiguration
metadata:
  annotations:
    manage.k8s.netris.ai/calico: "true"
    ...
  name: default
  ...
spec:
  asNumber: 64512
  nodeToNodeMeshEnabled: false
```

Note Netris Operator won't disable Node-to-Node mesh until k8s cluster all nodes' BGP peers are being established.

Finally, let's check if our earlier deployed "Podinfo" application is still working when Calico Node-to-Node mesh is disabled:

```
curl 50.117.59.202
```

Yes, it works:


```
{
 "hostname": "podinfo-576d5bf6bd-mfpdt",
 "version": "6.0.0",
 "revision": "",
 "color": "#34577c",
 "logo":
"https://raw.githubusercontent.com/stefanprodan/podinfo/gh-pages/cuddle_clap.gif",
 "message": "greetings from podinfo v6.0.0",
 "goos": "linux",
 "goarch": "amd64",
 "runtime": "go1.16.5",
 "num_goroutine": "8",
 "num_cpu": "4"
}
```


3.4.1 Disabling Netris-Calico Integration

To disable "Netris-Calico" integration, delete the annotation from Calico's `bgpconfigurations` resource:

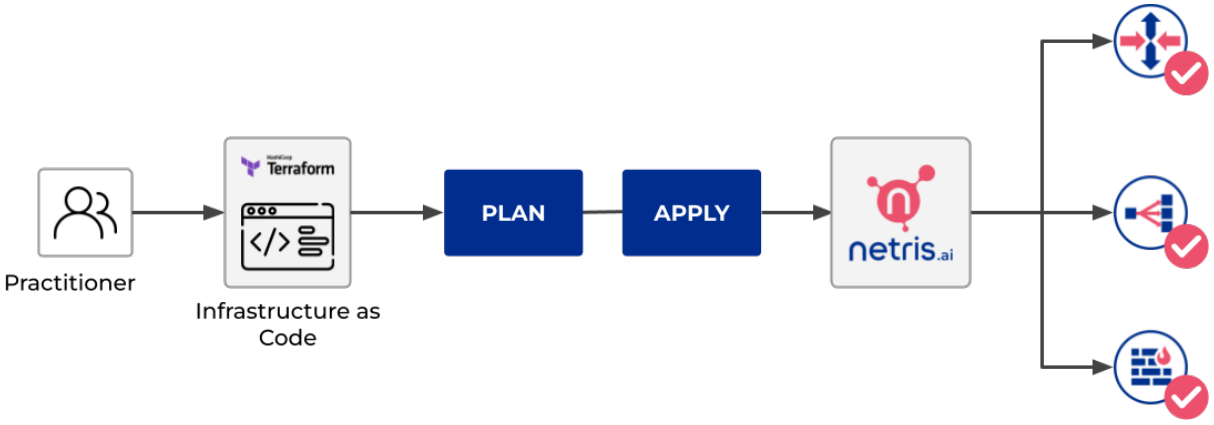
```
kubectl annotate bgpconfigurations default manage.k8s.netris.ai/calico-
```

or change its value to `"false"`.

Chapter 4

Terraform: Netris provider

Use Netris provider to interact with the many resources supported by Netris. You must configure the provider with the proper credentials before you can use it. To learn the basics of Terraform using this provider, follow the hands-on get started tutorials on HashiCorp's Learn platform.



When you make changes in the Terraform files and apply them, Terraform automatically decides which part of your configuration is already deployed into Netris controller and what should be added or removed.

To create your first Terraform configuration:

4.1 Install Terraform

Download and install the Terraform

4.2 Create a directory for Terraform files

1. Create a directory with any name, for example, `netris-terraform`. It stores the configuration files and saved states for Terraform and your infrastructure.
2. Create a configuration file with the `.tf` extension in this directory, such as `main.tf`.

4.3 Configure a provider

1. At the beginning of the configuration file, specify the provider settings.

```
terraform {
  required_providers {
    netris = {
      source  = "netrisai/netris"
    }
  }
}

provider "netris" {
  address = "<controller address>"
  login = "<controller login>"
  password = "<controller password>"
}
```

Specify the provider required arguments:

- `address` - This is your Netris-Controller address (http://example.com). This can also be specified with the `NETRIS_ADDRESS` environment variable.
- `login` - This is your Netris-Controller login. This can also be specified with the `NETRIS_LOGIN` environment variable.
- `password` - This is your Netris-Controller password. This can also be specified with the `NETRIS_PASSWORD` environment variable.

2. Execute the command `terraform init` in the folder with the configuration file. This command initializes the providers specified in the configuration files and lets you work with the provider resources and data sources.

4.4 Prepare an infrastructure plan

By using Netris Terraform Provider, you can create all kinds of resources, such as Sites, IPAMs, Topology, Inventory, etc. To create a resource, specify a set of required and optional parameters that define the resource properties. Such resource descriptions make up an infrastructure plan.

Infrastructure provisioning in Netris starts with Site resources. The Netris-Controller comes with the initial site `Default`. You can use it in your Terraform configuration files by getting its ID with the Terraform Data source element.

Let's create a separate file for site resource, and get its ID via Terraform Data source element.

```
cat << EOF > site.tf
data "netris_site" "default" {
  name = "Default"
}
EOF
```

Or, you can create a new `Site` resource, here is the detailed documentation with examples.

Now, when we're clear on `Site` resource usage, let's define our IPAM. There are two types of IPAM resources in the Netris-Controller it's `Allocation` and `Subnet`. IPAM resources only require `tenantid` field, let's get our default Admin tenant ID with the Data source element.

```
cat << EOF > tenant.tf
```

```
data "netris_tenant" "admin"{
  name = "Admin"
}
EOF
```

Then, when we have the `tenantid`, we can create IPAM resources.

```
cat << EOF > ipam.tf
resource "netris_allocation" "my-allocation-mgmt" {
  name = "my-allocation-mgmt"
  prefix = "192.0.2.0/24"
  tenantid = data.netris_tenant.admin.id
}

resource "netris_allocation" "my-allocation-loopback" {
  name = "my-allocation-loopback"
  prefix = "198.51.100.0/24"
  tenantid = data.netris_tenant.admin.id
}

resource "netris_allocation" "my-allocation-common" {
  name = "my-allocation-common"
  prefix = "203.0.113.0/24"
  tenantid = data.netris_tenant.admin.id
}

resource "netris_subnet" "my-subnet-mgmt" {
  name = "my-subnet-mgmt"
  prefix = "192.0.2.0/24"
  tenantid = data.netris_tenant.admin.id
  purpose = "management"
  defaultgateway = "192.0.2.254"
  siteids = [data.netris_site.default.id]
  depends_on = [
    netris_allocation.my-allocation-mgmt,
  ]
}

resource "netris_subnet" "my-subnet-loopback" {
  name = "my-subnet-loopback"
  prefix = "198.51.100.0/24"
  tenantid = data.netris_tenant.admin.id
  purpose = "loopback"
  siteids = [data.netris_site.default.id]
  depends_on = [
    netris_allocation.my-allocation-loopback,
  ]
}

resource "netris_subnet" "my-subnet-common" {
  name = "my-subnet-common"
  prefix = "203.0.113.0/25"
  tenantid = data.netris_tenant.admin.id
  purpose = "common"
  siteids = [data.netris_site.default.id]
  depends_on = [
    netris_allocation.my-allocation-common,
  ]
}
EOF
```

With the command above, we've defined 6 resources, 3 of the type of Allocation, 3 of the type of Subnet, each Subnet resource has a different purpose. For more details, get familiar with the IPAM docs.

Now, when we have all the required resources let's define our Inventory. We're going to create 1 SoftGate, 1 switch and connect them with a link.

```
cat << EOF > inventory.tf
resource "netris_softgate" "my-softgate" {
  name = "my-softgate"
  tenantid = data.netris_tenant.admin.id
  siteid = data.netris_site.default.id
  description = "Softgate 1"
  mainip = "auto"
  mgmtip = "auto"
  depends_on = [
    netris_subnet.my-subnet-mgmt,
    netris_subnet.my-subnet-loopback,
  ]
}

resource "netris_switch" "my-switch" {
  name = "my-switch"
  tenantid = data.netris_tenant.admin.id
  siteid = data.netris_site.default.id
  description = "Switch 01"
  nos = "cumulus_linux"
  asnumber = "auto"
  mainip = "auto"
  mgmtip = "auto"
  portcount = 16
  depends_on = [
    netris_subnet.my-subnet-mgmt,
    netris_subnet.my-subnet-loopback,
  ]
}

resource "netris_link" "sg-to-sw" {
  ports = [
    "swp1@my-softgate",
    "swp16@my-switch"
  ]
  depends_on = [
    netris_softgate.my-softgate,
    netris_switch.my-switch,
  ]
}
EOF
```

Next, let's define a local L3 network for our servers, suppose we want to connect 3 servers to our switch first 3 ports

```
cat << EOF > vnet.tf
resource "netris_vnet" "my-vnet" {
  name = "my-vnet"
  tenantid = data.netris_tenant.admin.id
  state = "active"
  sites{
    id = data.netris_site.default.id
    gateways {
      prefix = "203.0.113.1/25"
    }
    ports {
      name = "swp1@my-switch"
      vlanid = 1050
    }
    ports {
```

```
      name = "swp2@my-switch"
      vlanid = 1050
    }
    ports {
      name = "swp3@my-switch"
    }
  }
  depends_on = [
    netris_switch.my-switch,
    netris_subnet.my-subnet-common,
  ]
}
EOF
```

And finally, we have to provide internet connectivity to our fabric, for that we'll define BGP resource. Suppose we're going to connect our ISP cable to the 10th port of our switch, and want to establish the BGP session on our Softgate.

```
cat << EOF > bgp.tf
data "netris_port" "swp10_my_switch"{
  name = "swp10@my-switch"
  depends_on = [netris_switch.my-switch]
}

resource "netris_bgp" "my-bgp" {
  name = "my-bgp"
  siteid = data.netris_site.default.id
  hardware = "my-softgate"
  neighboras = 23456
  portid = data.netris_port.swp10_my_switch.id
  vlanid = 3000
  localip = "172.16.0.2/30"
  remoteip = "172.16.0.1/30"
  description = "My First BGP"
  prefixlistinbound = ["deny 127.0.0.0/8 le 32", "permit 0.0.0.0/0 le 24"]
  prefixlistoutbound = ["permit 192.0.2.0/24", "permit 198.51.100.0/24 le 25",
"permit 203.0.113.0/24 le 26"]
  depends_on = [netris_link.sg-to-sw]
}
EOF
```

Note For more information about all resources, how to create and manage them in Terraform, see the provider's documentation.

Now, when we've done with the configuration files, let's check whether they are valid

```
terraform validate
```

If the configuration is valid, the following message is returned:

```
Success! The configuration is valid.
```

4.5 Create resources

1. After preparing and checking the configuration, run the command:

```
terraform plan
```

The terminal will display a list of resources with parameters. This is a test step. No resources are created. If there are errors in the configuration, Terraform points them out.

2. To create resources, run the command:

```
terraform apply
```

3. Confirm the resource creation: type `yes` in the terminal and press **Enter**.

Terraform will create all the required resources and the terminal will display the progress. After creation, you can check resource availability and their settings in the Netris-Controller UI.

4.6 Delete resources

1. To delete resources created using Terraform:

Run the command:

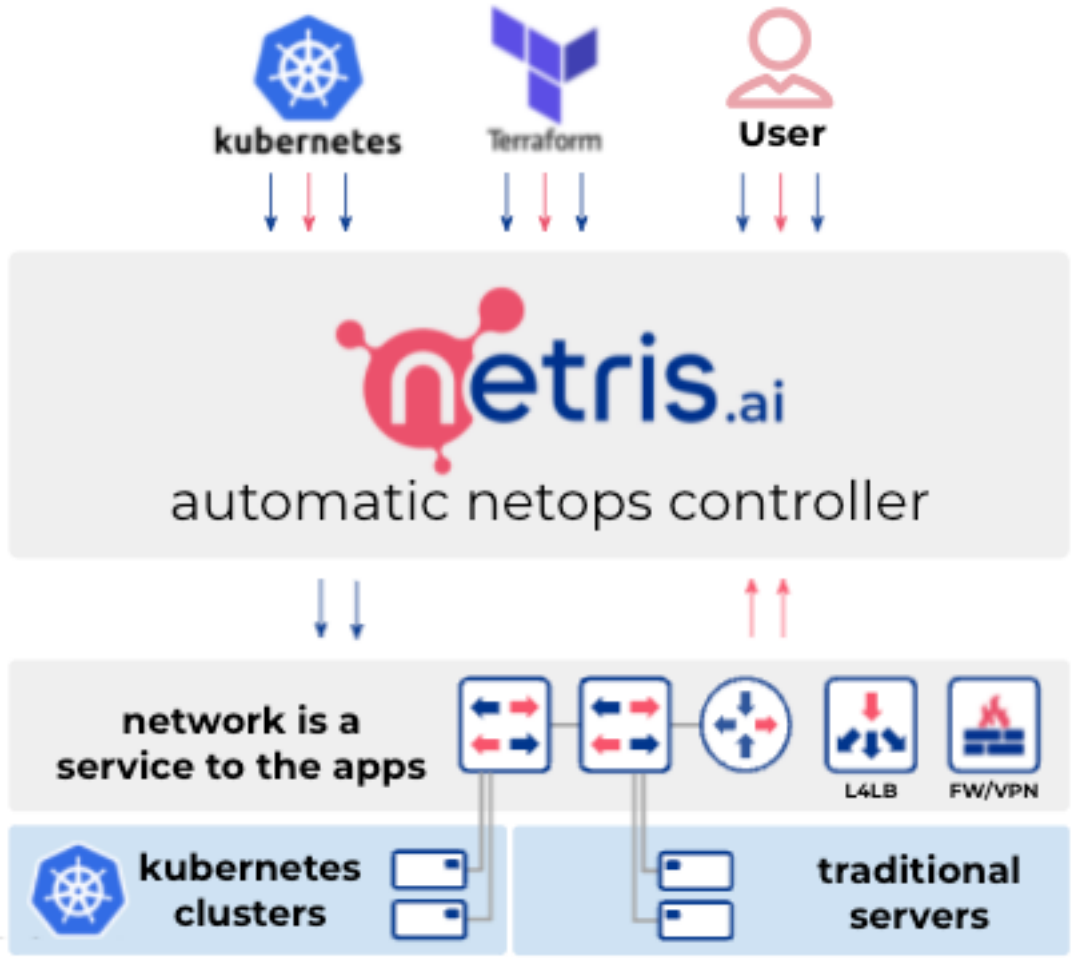
```
terraform destroy
```

After the command is executed, the terminal will display a list of resources to be deleted.

2. Type `yes` to confirm their deletion and press **Enter**.

Introduction to Netris

Netris is an automatic netops software for operating physical networks like it is a cloud. Netris automatically configures switching, routing, load-balancing, and network security based on user-defined services and policies. Netris continuously monitors the network's health and either applies software remediation or informs you of necessary actions if human intervention is required. Netris abstracts away the complexities of detailed network configuration, letting you perform efficiently by operating your physical network in a top down approach like a cloud – instead of the legacy box by box operation.



Netris Architecture

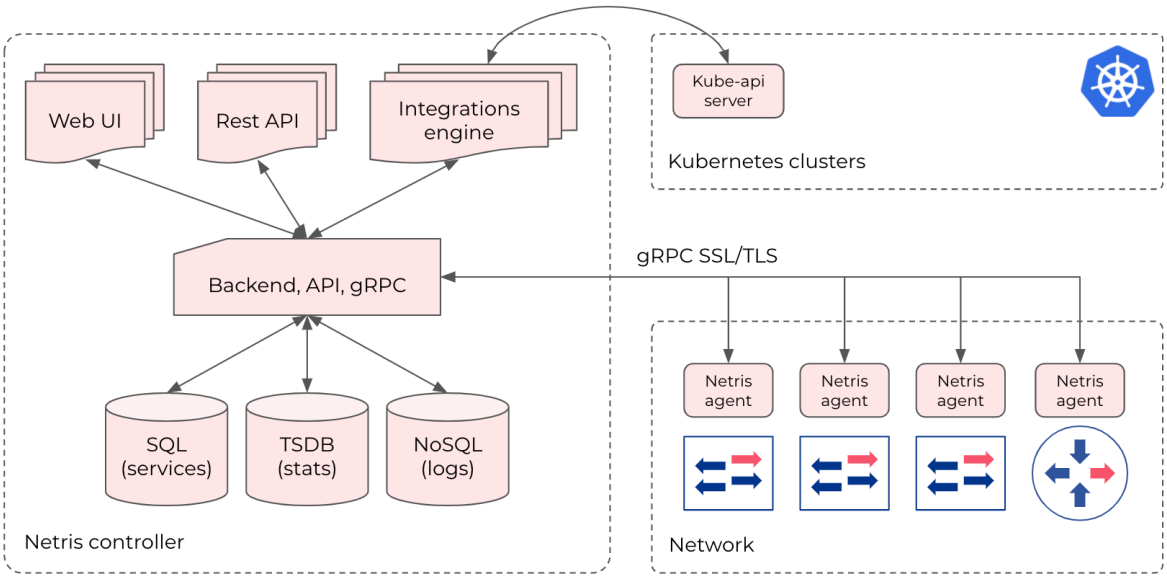
A Netris system is composed of 4 elements:

- Netris Controller
- Netris Switch Agent
- Netris SoftGate
- Customer Application Servers

6.1 Netris Controller

Netris Controller is the main operations control center for engineers using GUI/RestAPI/Kubernetes, systems, and network devices. The Netris Controller stores the data representing the user-defined network services and policies, health, statistics, analytics received from the network devices, and information from integration modules with external systems (Kubernetes). Netris Controller can run as a VM or container, on/off-prem, or in Netris cloud.

Diagram: High level Netris architecture



- **Controller HA** We highly recommend running more than one copy of the controller for database replication.
- **Multiple sites** Netris is designed to operate multiple sites with just a single controller with HA.

- **What if the controller is unreachable.** Netris operated switches/routers can tolerate the unreachability of the Netris Controller. Changes and stats collection will be unavailable during the controller unavailability window; however, switches/routers core operations will not be affected.

6.2 Netris Switch Agent

Netris Switch Agent is software running in the user space of the network operating system (NOS) of the switch and is responsible for automatically generating the particular switch configuration according to service requirements and policies defined in the Netris Controller. Netris Switch Agent uses an encrypted GRPC protocol for secure communication with the Netris Controller accessible through a local management network or over the Internet.

6.3 Netris SoftGate

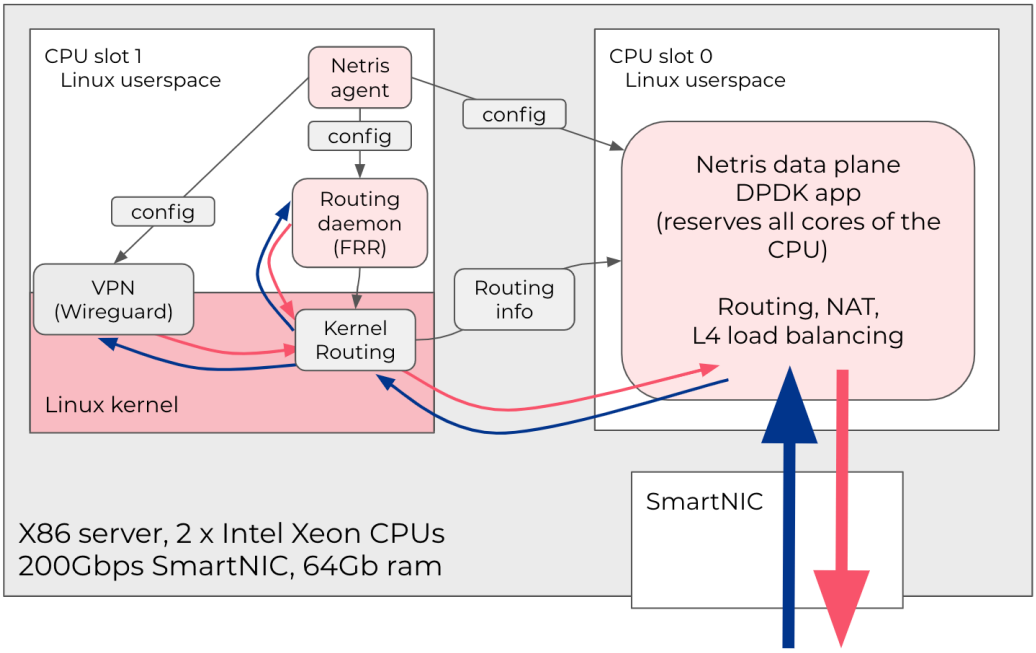
Netris SoftGate is automatic configuration software and reference architecture for enabling border routing, Layer-4 Load Balancing, Network Address Translation (NAT), and site-to-site VPN function on a regular x86 server with a SmartNIC card.

Netris SoftGate supports a high-performance DPDK data plane running in the user-space. It configures the system so that packets entering the NIC (network interface card) bypass Linux Kernel and go directly to the user space application. So traffic from the NIC travels through the PCIe bus to the closest CPU's last level cache and then into one of 8 cores, all reserved for the data-plane application. DPDK data-plane software processes the traffic for routing, load-balancing, NAT and makes necessary changes in the packet header (rewrites mac/VLAN-id) then returns the packet to the NIC, which sends it further into the switch for traveling further in Layer-2.

The server has to have 2 x Intel CPUs (8+ cores each). One CPU (closest to the SmartNIC card) is reserved for the data-plane process only (OS will report 100% CPU usage). Another CPU is used for running Linux OS, routing control plane (FRR), Netris agent, and other standard Linux utilities.

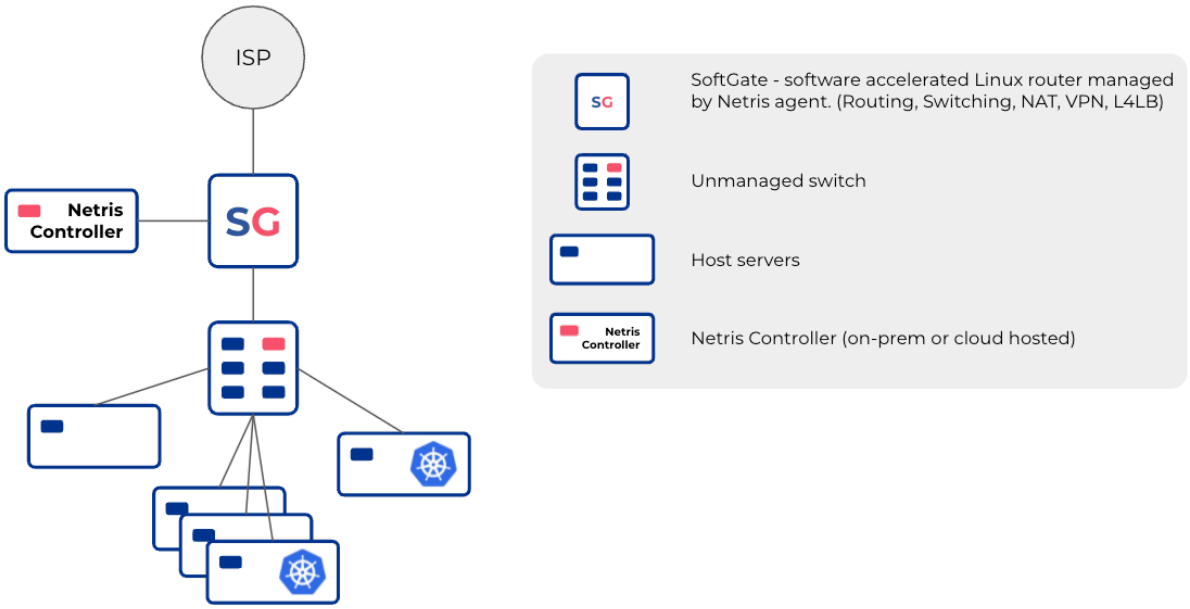
Netris agents can also configure Wireguard to form full mesh VPN tunnels between customer sites and then run necessary dynamic routing. So, servers and applications in multiple data centers can communicate over the Internet using encrypted tunnels.

Diagram: Netris SoftGate high level architecture

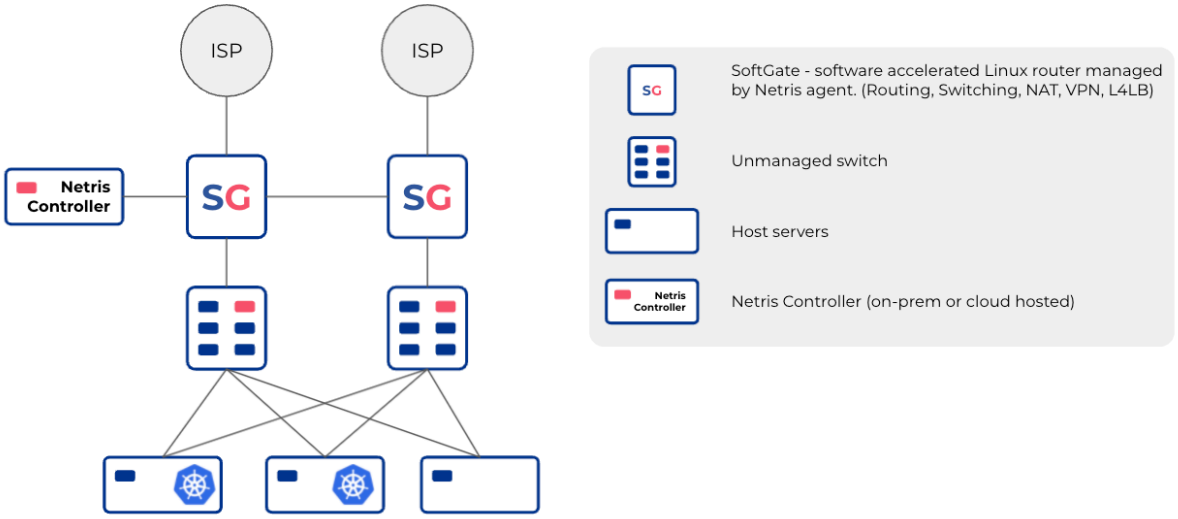

CPU slot 1
Linux userspace
Netris agent
config
config
Routing daemon (FRR)
config
VPN (Wireguard)
Kernel Routing
Linux kernel
Routing info
CPU slot 0
Linux userspace
Netris data plane DPDK app (reserves all cores of the CPU)
Routing, NAT, L4 load balancing
SmartNIC
X86 server, 2 x Intel Xeon CPUs
200Gbps SmartNIC, 64Gb ram

Reference Network Architectures

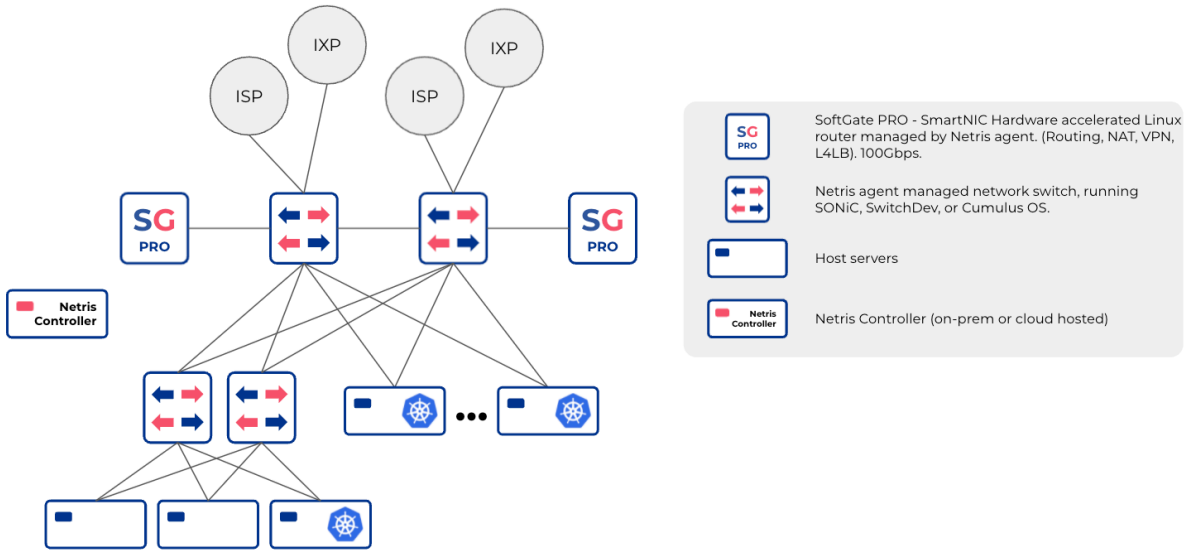
7.1 Unmanaged Switch & Netris SoftGate



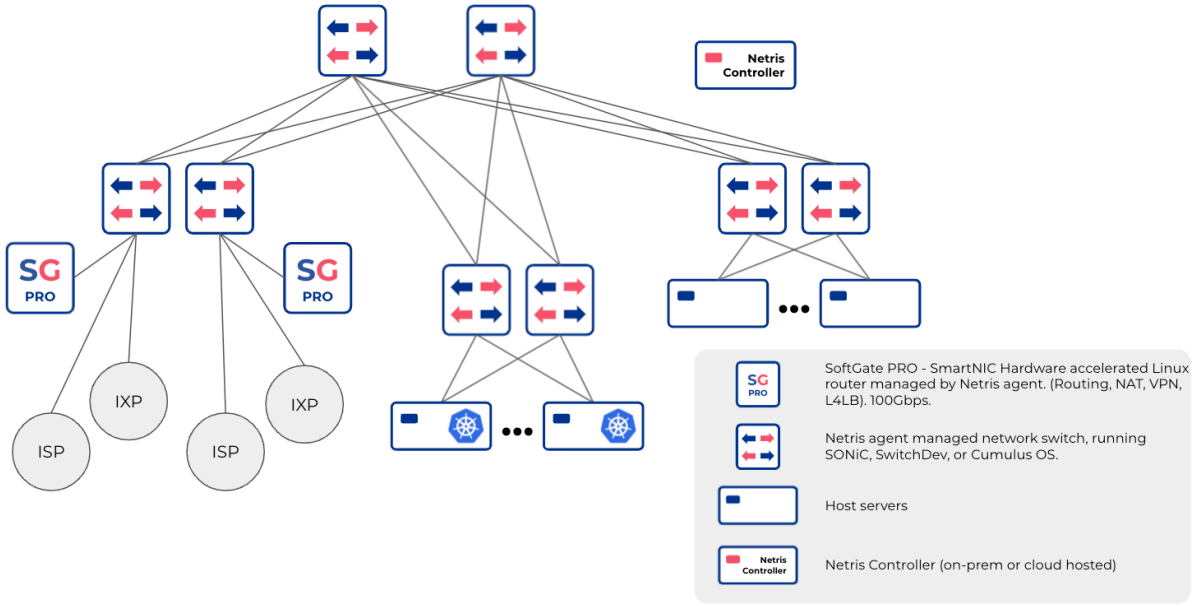
7.2 Unmanaged Switch & SoftGate (HA)



7.3 Netris Managed Switch & SoftGate (HA)



7.4 Netris Managed Switch & SoftGate scalable data center (HA)



Controller Installation

Netris Controller can be installed locally as a VM, deployed as a Kubernetes application, or hosted in the Netris Cloud. All three options provide the same functionality. Cloud-hosted Controller can be moved into on-prem anytime.

Note Select ONE installation option below.

8.1 Netris Controller installation on a generic Linux host

8.1.1 Linux Host requirements

- RAM: 8 GB
- CPU: 4 Cores
- Disk: 50GB
- OS: Linux 64-bit

Note K3s is expected to work on most modern Linux systems.

Some OSs have specific requirements:

- If you are using Raspbian Buster, follow these steps to switch to legacy iptables.
- If you are using Alpine Linux, follow these steps for additional setup.
- If you are using (Red Hat/CentOS) Enterprise Linux, follow these steps for additional setup.

8.1.2 Installation

The following command will install the Netris Controller on your Linux server:

```
curl -sfL https://get.netris.ai | sh -
```

Once installed, you will be able to log in to Netris Controller using your host's IP address.

Note The installation script does the following:

- Installs k3s
- Installs the Cert-Manager Helm chart
- Installs the Netris Controller Helm chart

Installation with the specific host name

In order to set the specific ingress host name to the Netris Controller, use the `--ctl-hostname` installation argument:

```
curl -sfL https://get.netris.ai | sh -s -- --ctl-hostname netris.example.com
```

A self-signed SSL certificate will be generated from that host name.

Installation with the Let's Encrypt SSL

The installation script supports Let's Encrypt SSL generation out-of-box. To instruct the installation script to do that use `--ctl-ssl-issuer` argument.

Note The argument `--ctl-ssl-issuer` is passing `cert-manager.io/cluster-issuer` value to the ingress resource of the Netris Controller. The installation script creates 2 resources type of ClusterIssuer: `selfsigned` and `letsencrypt`, where `selfsigned` is just Cert-Manager self-signed SSL and the `letsencrypt` is the ACME Issuer with HTTP01 challenge validation.

When `--ctl-ssl-issuer` isn't set installation script is proceeding with `selfsigned` ClusterIssuer.

Run the following command to install Netris Controller and use `letsencrypt` ClusterIssuer for SSL generation:

```
curl -sfL https://get.netris.ai | sh -s -- --ctl-hostname netris.example.com
--ctl-ssl-issuer letsencrypt
```

Note To successfully validate and complete Let's Encrypt SSL generation, a valid A/CNAME record for the domain/subdomain name should exist prior, and that name must be accessible from the Internet.

Installation with the Custom SSL Issuer

The HTTP01 challenge validation is the simplest way of issuing the Let's Encrypt SSL, but it doesn't work when the host behind the FQDN isn't accessible from the public internet. The common approach of validating and completing Let's Encrypt SSL generation for private deployments is DNS01 challenge validation. If the `DNS01` doesn't work for you either, Cert-Manager supports a number of certificate issuers, get familiar with all types of issuers here.

In order to install Netris Controller with the custom SSL issuer, you need to run installation script with the specified host name:

```
curl -sfL https://get.netris.ai | sh -s -- --ctl-hostname netris.example.com
```

Once the installation has finished, create a yaml file with the `ClusterIssuer` resource, suitable for your requirements, and apply it:

```
kubectl apply -f my-cluster-issuer.yaml
```

Then rerun the installation script with the `--ctl-ssl-issuer` argument:

```
curl -sfL https://get.netris.ai | sh -s -- --ctl-ssl-issuer <Your ClusterIssuer
resource name>
```

8.1.3 Upgrading

To upgrade the Netris Controller to the latest version simply run the script:

```
curl -sfL https://get.netris.ai | sh -
```

If a newer version of Netris Controller is available, it'll be updated in a few minutes.

8.1.4 Uninstalling

To uninstall Netris Controller and K3s from a server node, run:

```
/usr/local/bin/k3s-uninstall.sh
```

8.1.5 Backup and Restore

Netris Controller stores all critical data in MariaDB. It's highly recommended to create a cronjob with `mysqldump`.

Backup

To take database snapshot run the following command:

```
kubectl -n netris-controller exec -it netris-controller-mariadb-0 -- bash -c
'mysqldump -u $MARIADB_USER -p${MARIADB_PASSWORD} $MARIADB_DATABASE' >
db-snapshot-$(date +%Y-%m-%d-%H-%M-%S).sql
```

After command execution, you can find `db-snapshot-YYYY-MM-DD-HH-MM-SS.sql` file in the current working directory.

Restore

In order to restore DB from a snapshot, follow these steps:

In this example the snapshot file name is db-snapshot.sql and it's located in the current working directory

1. Copy snapshot file to the MariaDB container:

```
kubectl -n netris-controller cp db-snapshot.sql
netris-controller-mariadb-0:/opt/db-snapshot.sql
```

2. Run the restore command:

```
kubectl -n netris-controller exec -it netris-controller-mariadb-0 -- bash -c
'mysql -u root -p${MARIADB_ROOT_PASSWORD} $MARIADB_DATABASE <
/opt/db-snapshot.sql'
```

8.2 Helm Chart Installation

8.2.1 Requirements

- Kubernetes 1.12+
- Helm 3.1+
- PV provisioner support in the underlying infrastructure

8.2.2 Get Repo Info

Add the Netris Helm repository:

```
helm repo add netrisai https://netrisai.github.io/charts
helm repo update
```

8.2.3 Installing the Chart

In order to install the Helm chart, you must follow these steps:

1. Create the namespace for netris-controller:

```
kubectl create namespace netris-controller
```

1. Install helm chart with netris-controller:

```
helm install netris-controller netrisai/netris-controller \
  --namespace netris-controller \
  --set app.ingress.hosts={my.domain.com}
```

8.2.4 Uninstalling the Chart

To uninstall/delete the `netris-controller` helm release:

```
helm uninstall netris-controller
```

8.2.5 Chart Configuration

See the netris-controller README for details about configurable parameters and their default values.

Chapter 9

Switch Agent Installation

9.1 Prerequisite Steps

9.1.1 Nvidia Cumulus Linux Devices

Requirements: * Fresh install of Cumulus Linux v. 3.7.(x) - Cumulus 4.X is in the process of validation and will be supported in the next Netris release.

Configure the OOB Management IP address

Configure internet connectivity via management port.

```
sudo vim /etc/network/interfaces
```

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address <management IP address/prefix length>
        gateway <gateway of management network>
        dns-nameserver <dns server>

source /etc/network/interfaces.d/*
```

```
sudo ifreload -a
```

Configure Nvidia Cumulus Linux License

```
sudo cl-license -i
```

Copy/paste the Cumulus Linux license string then press ctrl-d.

Continue to Section 9.2 section.

9.1.2 Ubuntu SwitchDev Devices

Note Further installation requires a Console and Internet connectivity via management port!

1. NOS Uninstall

Uninstall current NOS using **Uninstall OS** from grub menu:

Once the uninstallation is completed, the switch will reboot automatically.

2. Update ONIE

Select **Update ONIE** from grub menu:

In case you don't have DHCP in the management network, then stop ONIE discovery service and configure IP address and default gateway manually:

```
onie-discovery-stop
ip addr add <management IP address/prefix> dev eth0
ip route add default via <gateway of management network>
echo "nameserver <dns server>" > /etc/resolv.conf
```

Update ONIE to the supported version.

Note ONIE image available for Mellanox switches only!

```
onie-self-update http://downloads.netris.ai/onie-updater-x86_64-mlnx_x86-r0
```

3. NOS Install

Select **Install OS** from grub menu:

In case you don't have DHCP in the management network, then stop ONIE discovery service and configure IP address and default gateway manually:

```
onie-discovery-stop
ip addr add <management IP address/prefix> dev eth0
ip route add default via <gateway of management network>
echo "nameserver <dns server>" > /etc/resolv.conf
```

Install Ubuntu-SwitchDev from the Netris custom image:

```
onie-nos-install http://downloads.netris.ai/netris-ubuntu-18.04.1.bin
```

Default username/password

`netris/newNet0ps`

Configure the OOB Management IP address

Configure internet connectivity via management port.

```
sudo vim /etc/network/interfaces
```

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address <management IP address/prefix length>
        gateway <gateway of management network>
        dns-nameserver <dns server>

source /etc/network/interfaces.d/*
```

```
sudo ifreload -a
```

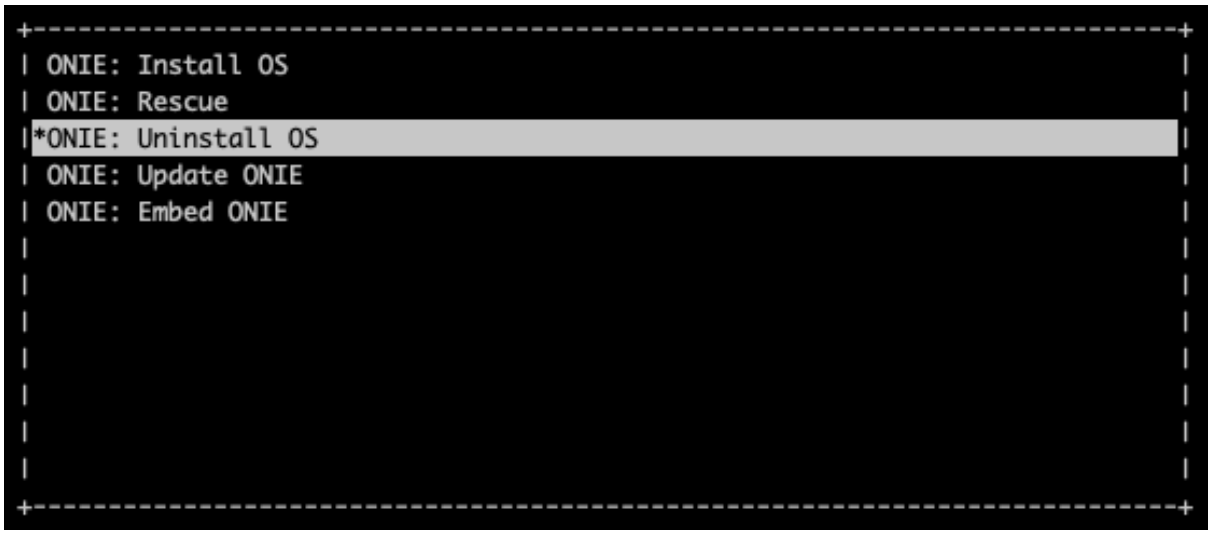
Continue to Section 9.2 section.

9.1.3 EdgeCore SONiC Devices

Note Further installation requires a Console and Internet connectivity via management port!

1. NOS Uninstall

Uninstall current NOS using **Uninstall OS** from grub menu:



Once the uninstallation is completed, the switch will reboot automatically.

2. NOS Install

Select **Install OS** from grub menu:

In case you don't have DHCP in the management network, then stop ONIE discovery service and configure IP address and default gateway manually:

```
onie-discovery-stop
ip addr add <management IP address/prefix> dev eth0
ip route add default via <gateway of management network>
echo "nameserver <dns server>" > /etc/resolv.conf
```

Install EdgeCore SONiC image from the Netris repository:

```
onie-nos-install
http://downloads.netris.ai/Edgecore-SONiC_20211125_074752_ec202012_227.bin
```

Default username/password

```
admin/YourPaSsWoRd
```

Configure the OOB Management IP address

Disable Zero Touch Provisioning for time being.

```
ztp disable -y
```

Note This will take some time, please be patient.

Configure internet connectivity via management port.

```
ip addr add <management IP address/prefix> dev eth0
ip route add default via <gateway of management network>
echo "nameserver <dns server>" > /etc/resolv.conf
```

Continue to Section 9.2 section.

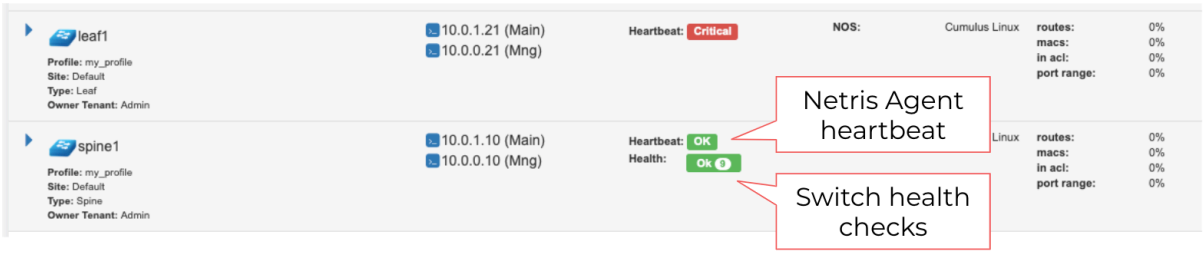
9.2 Install the Netris Agent

1. Add the Switch in the controller **Inventory**. Detailed configuration documentation is available here: Section 13.2
2. Once the Switch is created in the **Inventory**, click on **three vertical dots (?** on the right side on the Switch and select the **Install Agent** option
3. Copy the agent install command to your clipboard and run the command on the Switch
4. Reboot the Switch when the installation completes

```
sudo reboot
```

Once the switch boots up you should see its heartbeat going from Critical to OK in Net→Inventory, Telescope→Dashboard, and switch color will reflect its health in Net→Topology

Screenshot: Net→Inventory



SoftGate Agent Installation

10.1 Minimum Hardware Requirements

- 2 x Intel Silver CPU
- 96 GB RAM
- 300 GB HDD
- Nvidia Mellanox Connect-X 5/6 SmartNIC card

10.2 BIOS Configuration

The following are some recommendations for BIOS settings. Different vendors will have different BIOS naming so the following is mainly for reference:

- Before starting consider resetting all BIOS settings to their defaults
- Disable all power saving options such as: Power performance tuning, CPU P-State, CPU C3 Report and CPU C6 Report
- Select Performance as the CPU Power and Performance policy
- Enable Turbo Boost
- Set memory frequency to the highest available number, NOT auto
- Disable all virtualization options when you test the physical function of the NIC, and turn off VT-d
- Disable Hyper-Threading

10.3 Install the Netris Agent

Requires freshly installed Ubuntu Linux 18.04 and internet connectivity configured from netplan via management port

1. Add the SoftGate in the controller **Inventory**. Detailed configuration documentation is available here: Section 13.3
2. Once the SoftGate is created in the **Inventory**, click on **three vertical dots (?** on the right side on the SoftGate and select the **Install Agent** option
3. Copy the agent install command to your clipboard and run the command on the SoftGate

4. When the installation completes, review ifupdown configuration file (it was generated based on your netplan configuration)

```
sudo vim /etc/network/interfaces
```

5. If everything seems ok, remove/comment **Gateway** line and save the file.

Note If the Netris Controller is not in the same OOB network then add a route to Netris Controller. No default route or other IP addresses should be configured.

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
      address <Management IP address/prefix length>
      up ip ro add <Controller address> via <Management network gateway> #delete
this line if Netris Controller is located in the same network with the SoftGate
node.

source /etc/network/interfaces.d/*
```

6. Reboot the SoftGate

```
sudo reboot
```

Once the server boots up you should see its heartbeat going from Critical to OK in Net→Inventory, Telescope→Dashboard, and SoftGate color will reflect its health in Net→Topology

Definitions

When configuring and operating a Netris system, the following nomenclature is important to understand:

- **User** - A user account for accessing Netris Controller through GUI, RestAPI, and Kubernetes. The default username is `netris`, with password `newNet0ps`.
- **Tenant** - IP addresses and Switch Ports are network resources assigned to different Tenants to have under their management. Admin is the default tenant, and by default, it owns all the resources. You can use different Tenants for sharing and delegation of control over the network resources. Network teams typically use Tenants to grant access to other groups to request and manage network services using the Netris Controller as a self-service portal or programmatically (with Kubernetes CRDs) via a DevOps/NetOps pipeline.
- **Permission Group** - List of permissions on a per section basis can be attached individually to a User or a User Role
- **User Role** - Group of user permissions and tenants for role-based access control (RBAC)
- **Site** - Each separate deployment (each data center) should be defined as a *Site*. All network units and resources are attached to a site. Netris Controller comes with a "default" site preconfigured. Site entry defines global attributes such as; AS numbers, default ACL policy, and Site Mesh (site to site VPN) type.
- **Subnet** - IPv4/IPv6 address resources linked to *Sites* and *Tenants*
- **Switch Port** - Physical ports of all switches attached to the system
- **Inventory** - Inventory of all network units that are operated using Netris Agent
- **E-BGP** - Defines all External BGP peers (iBGP and eBGP)

Chapter 12

IP Address Management

Netris IPAM allows users to document their IP addresses and track pool usage. It is designed to have a tree-like view to provide opportunity to perform any kind of subnetting.

Purpose: Users define specific roles(purpose) for each subnet/address and only after that are allowed to use those subnets in services like V-net, NAT, etc…

12.1 Allocations and Subnets

There are 2 main types of IP prefixes - allocation and subnet. Allocations are IP ranges allocated to an organization via RIR/LIR or private IP ranges that are going to be used by the network. Subnets are prefixes which are going to be used in services. Subnets are always childs of allocation. Allocations do not have parent subnets.

12.2 Add an Allocation

1. Navigate to Net→IPAM
2. Click the **Add** button
3. Select **Allocation** from the bottom select box
4. Fill in the rest of the fields based on the requirements listed below
5. Click the **Add** button

Table 12.1. Allocation Fields

Name	Unique name for current allocation.
Prefix	Unique prefix for allocation, must not overlap with other allocations.
Tenant	Owner of the allocation.

12.3 Add a Subnet

1. Navigate to Net→IPAM
2. Click the **Add** button

3. Select **Subnet** from the bottom select box
4. Fill in the rest of the fields based on the requirements listed below
5. Click the **Add** button

Table 12.2. Subnet fields

Name	Unique name for current subnet.
Prefix	Unique prefix for subnet, ust be included in one of allocations.
Tenant	Owner of the subnet.
Purpose	This field describes for what kind of services the current subnet can be used. It can have the following values: • *common* - ordinary subnet, can be used in v-nets and ROH. • *loopback* - hosts of this subnet can be used only as loopback IP addresses for Netris hardware (switches and/or softgates). • *management* - subnet which specifies the out-of-band management IP addresses for Netris hardware (switches and softgates). • *load-balancer* - hosts of this subnet are used in L4LB services only. Useful for deploying on-prem kubernetes with cloud-like experience. • *nat* - hosts of this subnet or subnet itself can be used to define NAT services. • *inactive* - can't be used in any services, useful for reserving/documenting prefixes for future use.

Inventory

The Inventory section allows you to add/edit/delete network switches and SoftGates. Initial setup of a Netris managed network is a three part process:

1. Create Inventory Profiles
2. Adding Switches
3. Adding Softgates

13.1 Inventory Profiles

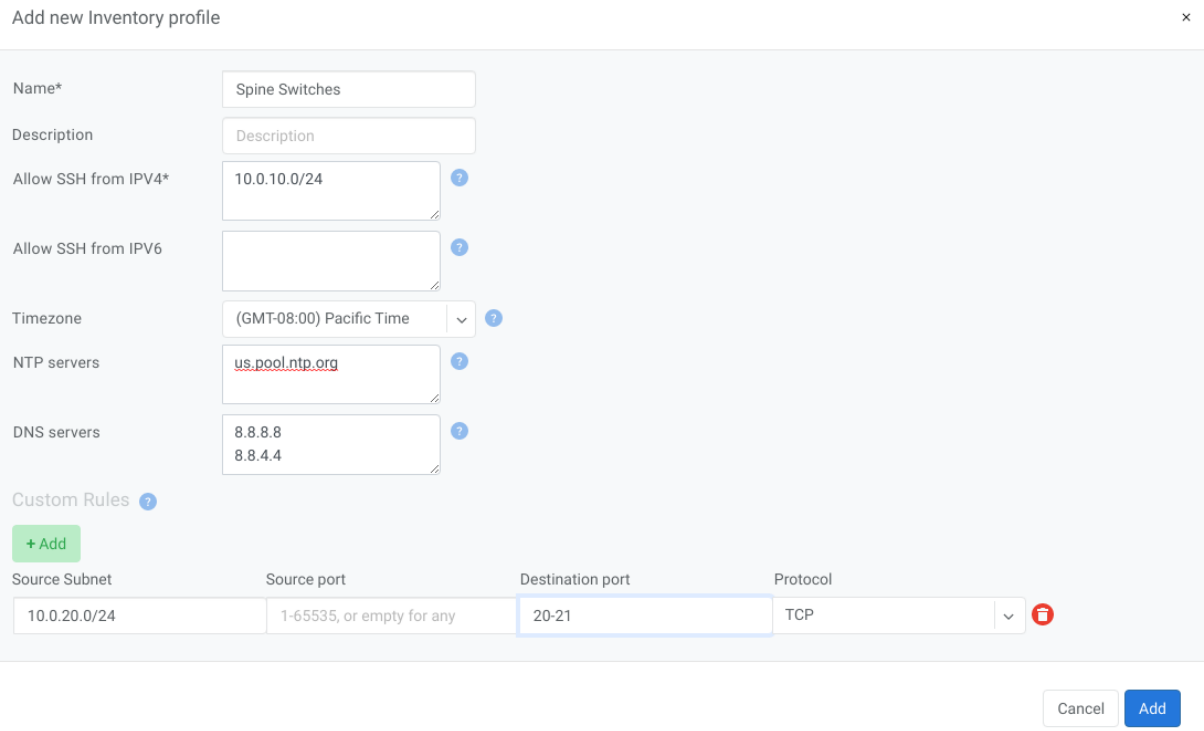
Inventory profiles allow security hardening of inventory devices. By default all traffic flow destined to switch/SoftGate is allowed. As soon as the inventory profile is attached to a device it denies all traffic destined to the device except Netris-defined and user-defined custom flows. Generated rules include:

- SSH from user defined subnets
- NTP from user defined ntp services
- DNS from user defined DNS servers
- Custom user defined rules

Table 13.1. Inventory Profile Fields

Name	Profile name
Description	Free text description
Allow SSH from IPv4	List of IPv4 subnets allowed to ssh (one address per line)
Allow SSH from IPv6	List of IPv6 subnets allowed to ssh (one address per line)
Timezone	Devices using this inventory profile will adjust their system time to the selected timezone.
NTP servers	List of domain names or IP addresses of NTP servers (one address per line). You can use your Netris Controller address as an NTP server for your switches and SoftGate.
DNS servers	List of IP addresses of DNS servers (one address per line). You can use your Netris Controller address as a DNS server for your switches and SoftGate.

Example: In this example Netris Controller is used to provide NTP and DNS services to the switches (common setup).



13.2 Adding Switches

Every switch needs to be added to the Netris Controller inventory. You can add new devices with the following process:

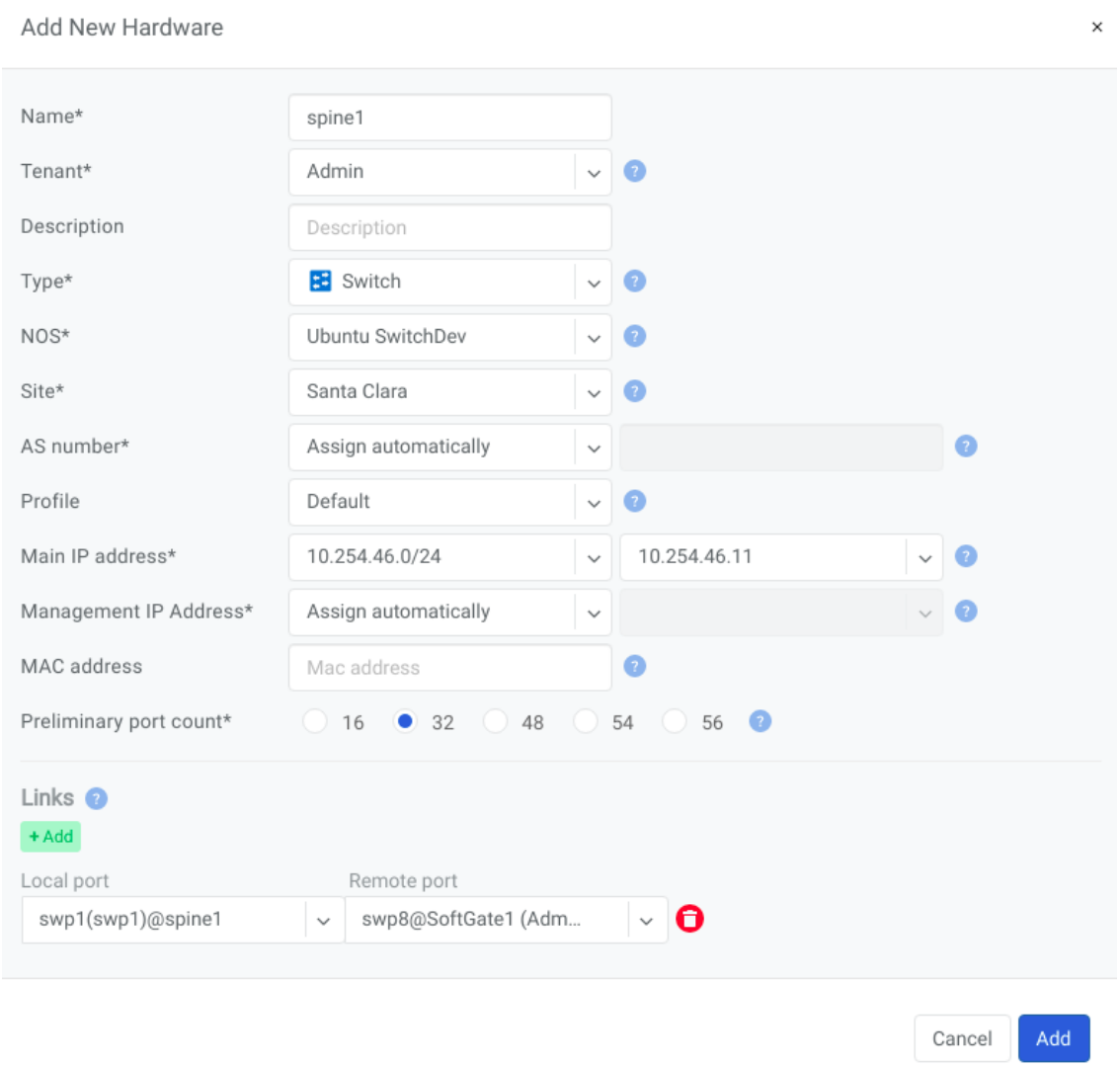
1. Navigate to **Net→Inventory**
2. Click the **Add** button
3. Fill in the fields as described below
4. Click the **Add** button

Table 13.2. Add Inventory Fields - Switch

Name	Name of the device
Owner Tenant	Owner tenant of the device
Description	Description of the device
Type	There are 3 types of devices that users can add/edit - Switch/Softgate/Controller. Other types are added automatically when creating services like ROH
NOS	Operating system of the device; applicable for switches only
Site	Site where the devices reside
AS Number	Private AS number of the device; applicable for switches only; recommended to be assigned automatically
Profile	Inventory profile for current device. Profiles are used for security hardening the devices
Main IP address	Main loopback IP address for the device. Can be configured manually or assigned automatically from subnet with *loopback* purpose defined for current site.

Management IP address	Management IP address for the device. Can be configured manually or assigned automatically from subnet with *management* purpose defined for current site. This IP address is configured on the out-of-band management interface of the device.
MAC address	MAC address of the device; applicable for switches only
Preliminary port count	Used for definition of topology. When the device registers with the controller the real ports are synced with inventory
Add Link	Provides functionality to define the connections between devices; mandatory for Switch and Softgate physical interconnections

Example: Add a new Switch.



Note Repeat this process to define all your switches.

13.3 Adding SoftGates

Every SoftGate node needs to be added to the Netris Controller inventory. To add a SoftGate node:

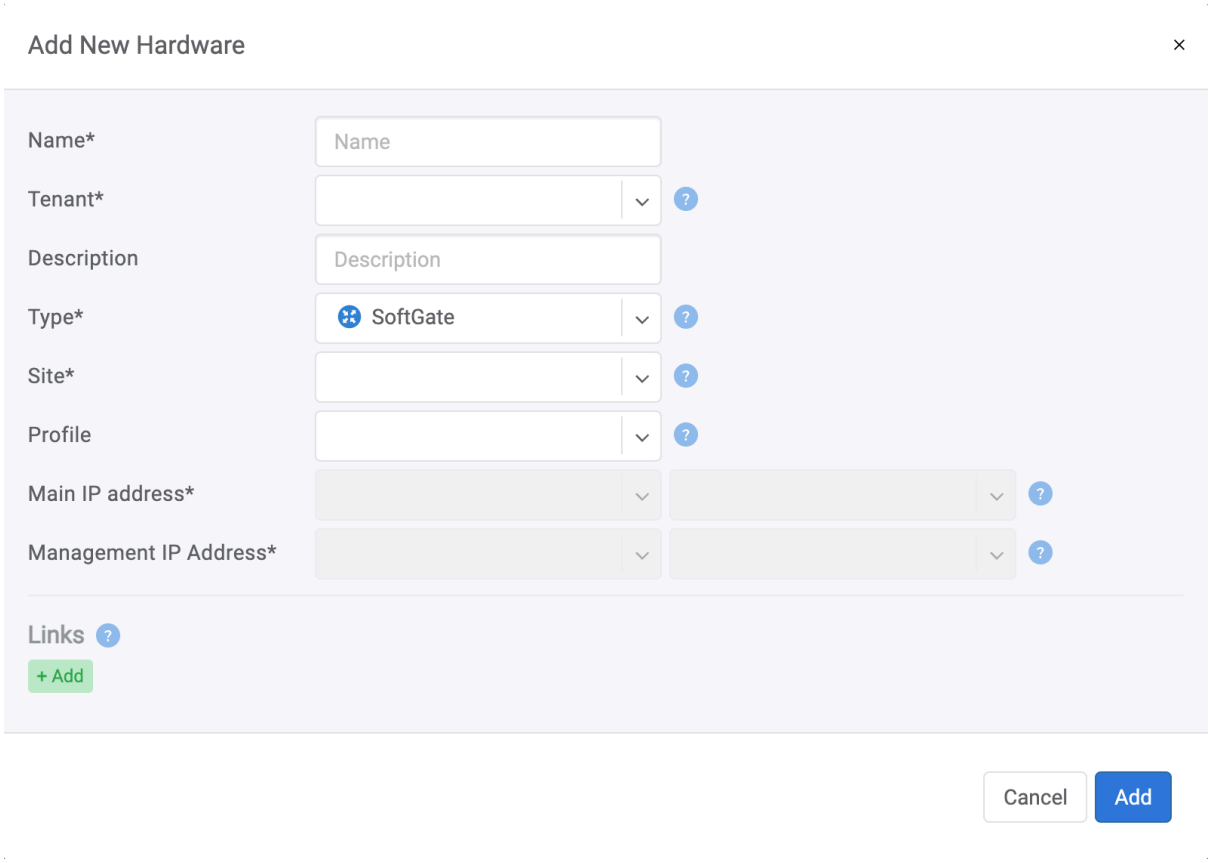
1. Navigate to **Net→Topology**

2. Click **Add**
3. Fill in the fields as described below
4. Click the **Add** button

Table 13.3. Add Inventory Fields - SoftGate

Name	Descriptive name
Owner Tenant	Tenant(typically Admin); who administers this node
Description	Free text description
Hardware Type	Select **SoftGate**
Site	The data center where the current SoftGate node belongs.
Inventory Profile	Profile describing the timezone; DNS; NTP; and Security features
IP Address	IPv4 address for the loopback interface
Management IP address	IPv4 address for the out of band management interface
NFV Node Port	A physical port on a spine switch where the SoftGate node's first SmartNIC port is connected. Typically each spine switch has one SoftGate node connected to it.
+NAT address	Public IP addresses to be used as global IP for SNAT/DNAT. (check Enabling NAT section of Network Policies chapter)
+NAT address pool	Public IP address subnets to be used as rolling global IP addresses for SNAT. (check Enabling NAT section of Network Policies chapter)

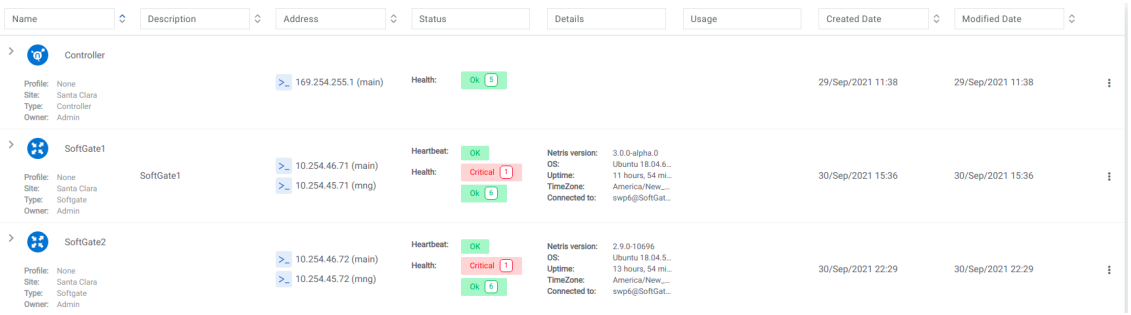
Example: Adding a SoftGate Node to Topology.



13.4 Viewing Inventory

Inventory Listing shows also Heartbeat and monitoring statuses of each device.

Heartbeat - Shows the status of device reachability. Health - Shows number of successful and failed checks on the device.



Note You can also add new devices in the Topology view.

Topology Manager

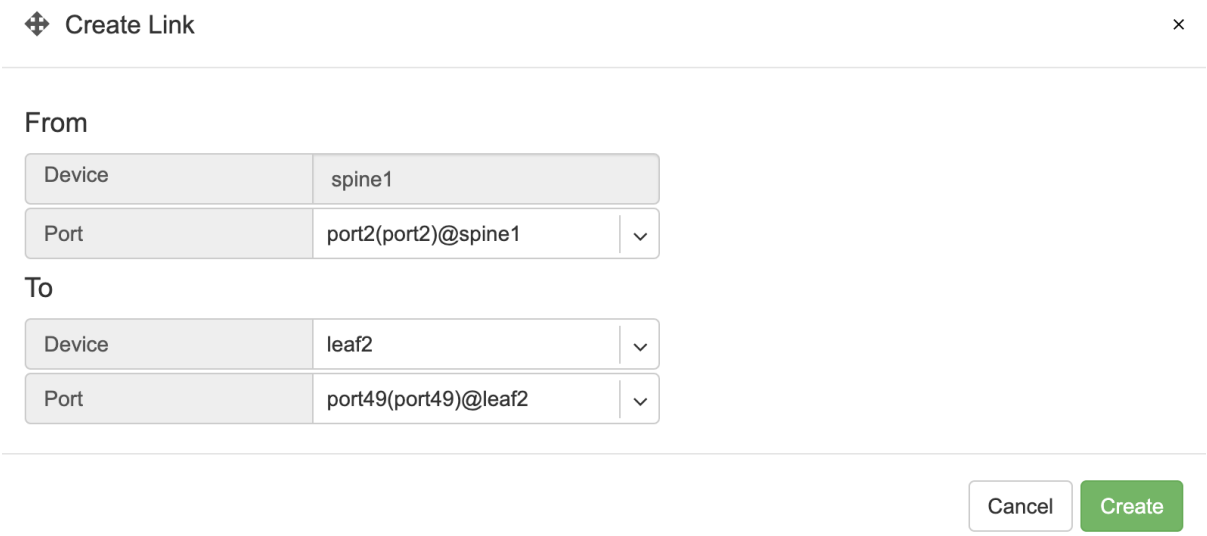
The topology manager is for describing and monitoring the desired network topology. Netris Switch Agents will configure the underlying network devices according to this topology dynamically and start watching against potential failures.

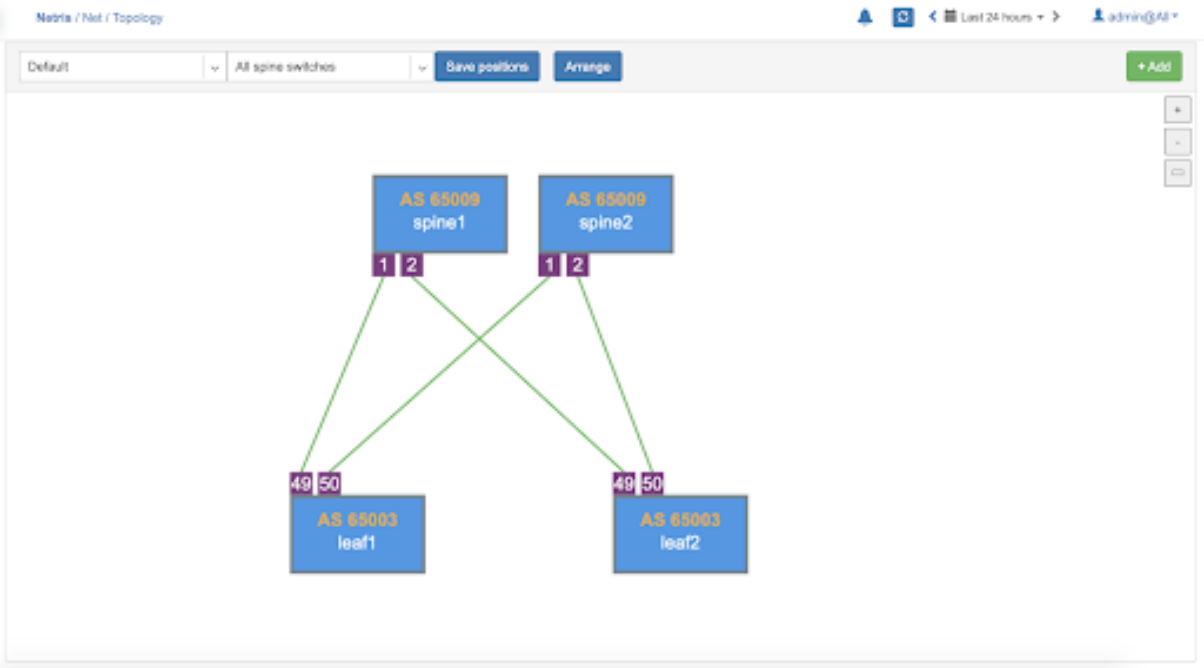
14.1 Adding Links

To define the links in the network:

1. Right-click on the spine switch
2. Click **Create Link**
3. Select the **From Port** and the **To Port**

See the example below:





Once the links have been defined, the network is automatically configured as long as physical connectivity is in place and Netris Agents can communicate with Netris Controller.

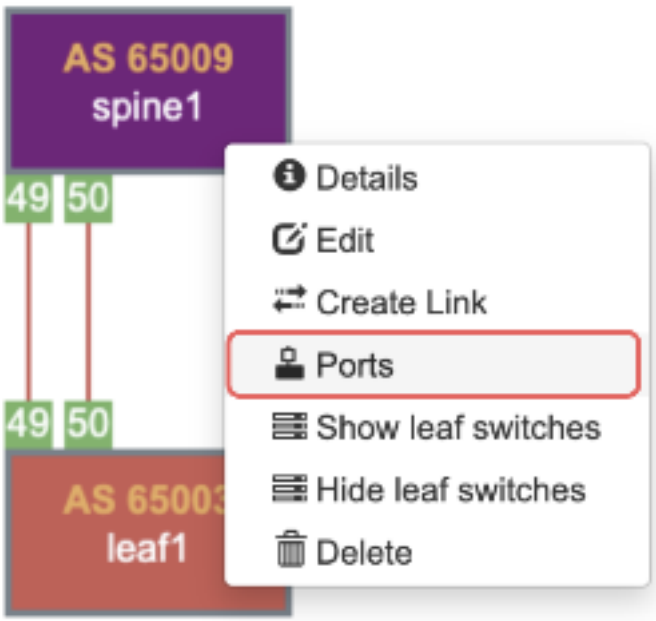
Tip You can drag/move the units to your desired positions and click "Save positions".

14.2 Hairpin Links (Nvidia Cumulus only)

With Nvidia Cumulus Linux only, we need to loop two ports on spine switches (hairpin cable) in the current release, usually two upstream (higher capacity) ports. We are planning to lift this requirement in the next Netris release (v2.10).

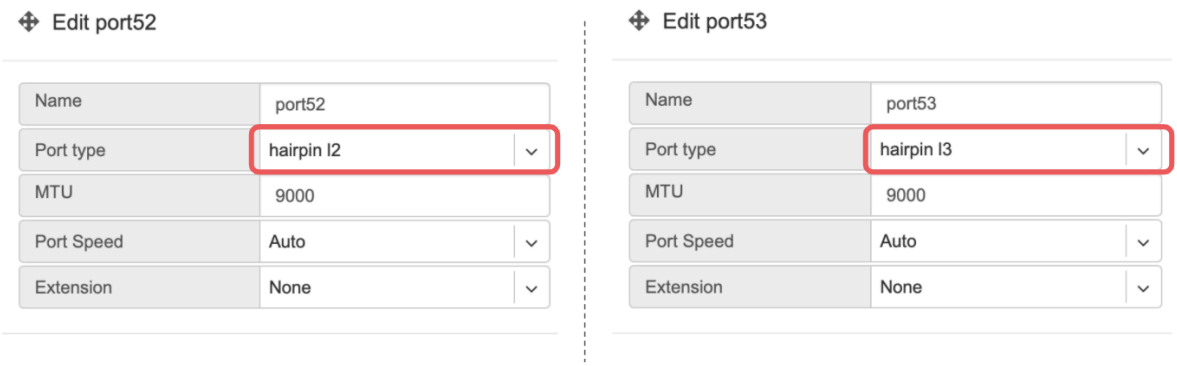
To define what ports will be used as a hairpin, navigate to Net→Switch Ports, or right-click on the spine switch, click Ports in Net–>Topology.

Example: Accessing Switch Ports from Net→Topology

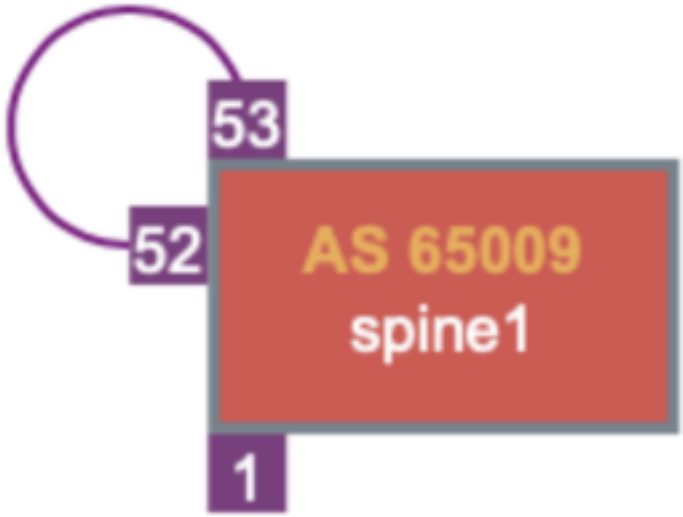


For each spine switch, find the two ports that you are going to connect (loop/hairpin) and configure one port as a "hairpin **l2**" and another port as "hairpin **l3**". The order doesn't matter. The system needs to know which ports you have dedicated for the hairpin/loop on each spine switch. (do not do this for non-Cumulus switches) | Example: Editing Switch Port from Net→Switch Ports.

Example: Setting port types to "hairpin l2" and "hairpin l3".



Screenshot: Hairpin visualized in Net→Topology

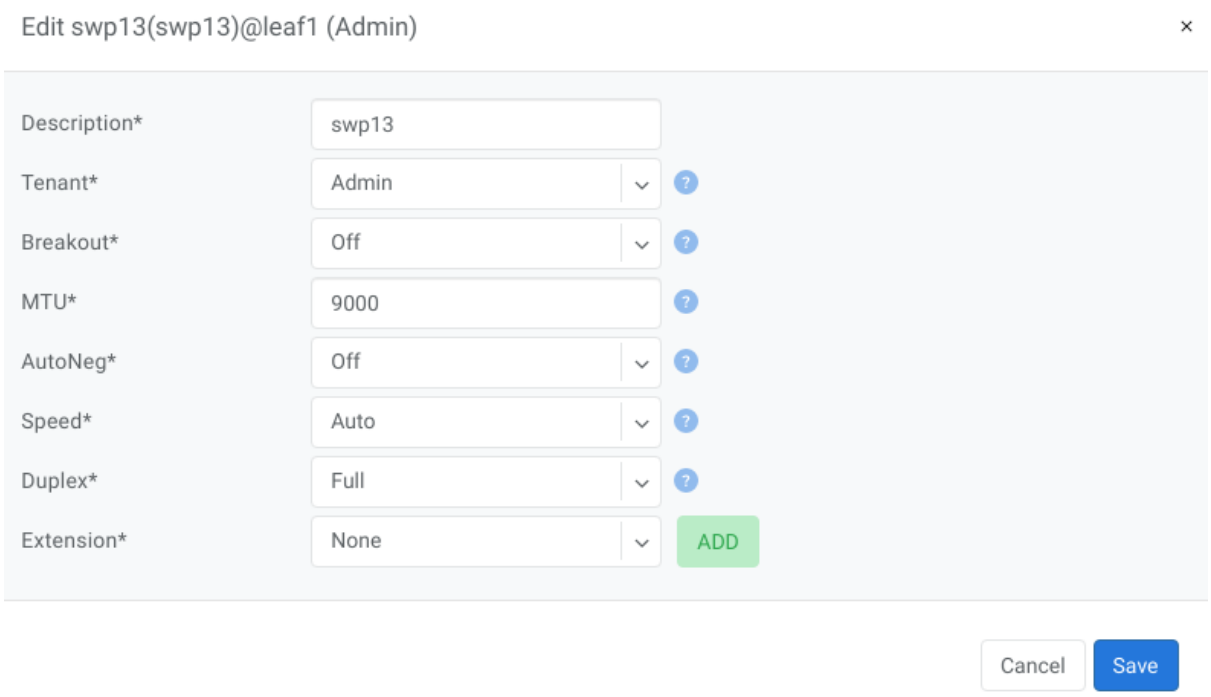

53
52
AS 65009
spine1
1

Switch Ports

Switch ports can be directly managed in the **Switch Port** UI section. Both physical and virtual ports (extended, aggregate, etc…) will appear in this section once they have been added to inventory. The Netris Controller will automatically sync the list of available ports that appear on each device.

The following options are available for editing on each port:

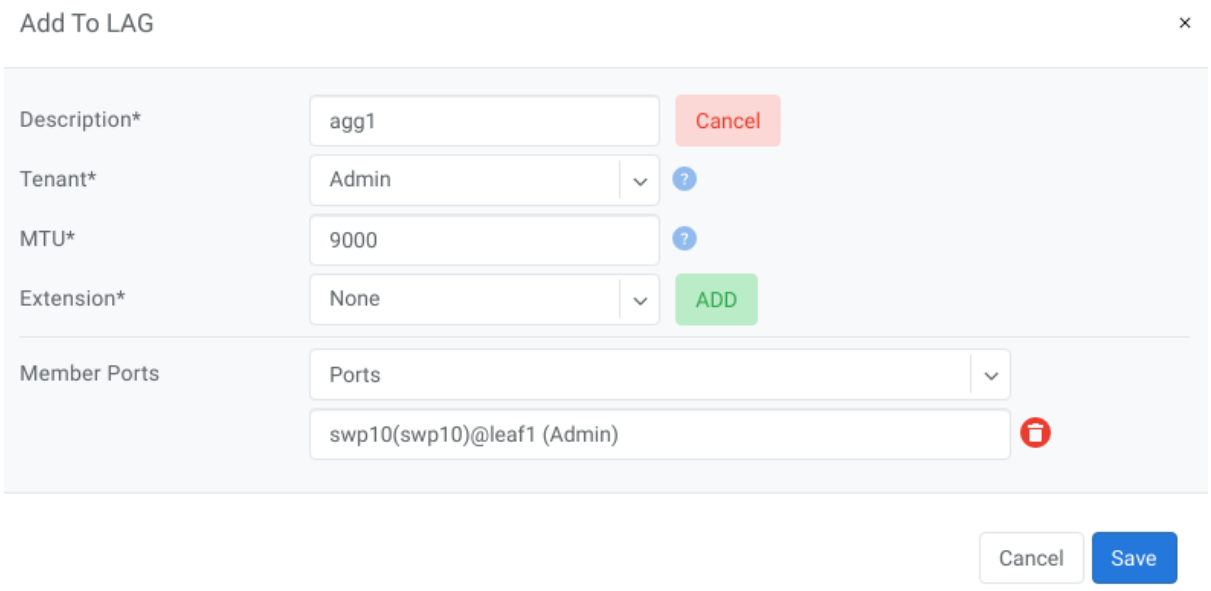
- Description - Description of the port.
- Tenant - Tenant to whom the port is assigned, by default it is the owner tenant of the device to whom the port belongs to.
- Breakout - Available only for physical switch ports, used to split physical ports into multiple physical ports. When there is a need to use other supported option supported by switch not shown in the dropdown list user must set breakout to “Manual” and configure breakout manually on the switch. For certain platforms some ports need to be disabled to support breakout into other ports, for that option use “Disable” mode of breakout. For Cumulus, after configuration, user must manually restart switchd daemon on the switch via command “systemctl restart switchd”.
- MTU - Maximum transmission unit of the port.
- Autoneg - Toggle autonegotiation. Available only for physical ports.
- Speed - Toggle speed. Available only for physical ports.
- Duplex - Toggle duplex. Available only for physical ports.
- Extension - Create extension ports. Available for physical and aggregate ports.
- Extension Name - Name for new extension.
- VLAN Range - VLAN id range for new extension port.



Example: Edit physical port

Quick action menu provides following actions for ports (note that Bulk Action also available for multiple ports:

Edit - Edit the port. Admin UP/Down - Toggle admin status of the port. Add to V-net - Add selected port(s) to a V-net.



Free Up Port - Detach port from all resources.

Chapter 16

Basic BGP

BGP neighbors can be declared in the Net→E-BGP section. Netris will automatically generate and program the network configuration to meet the requirements.

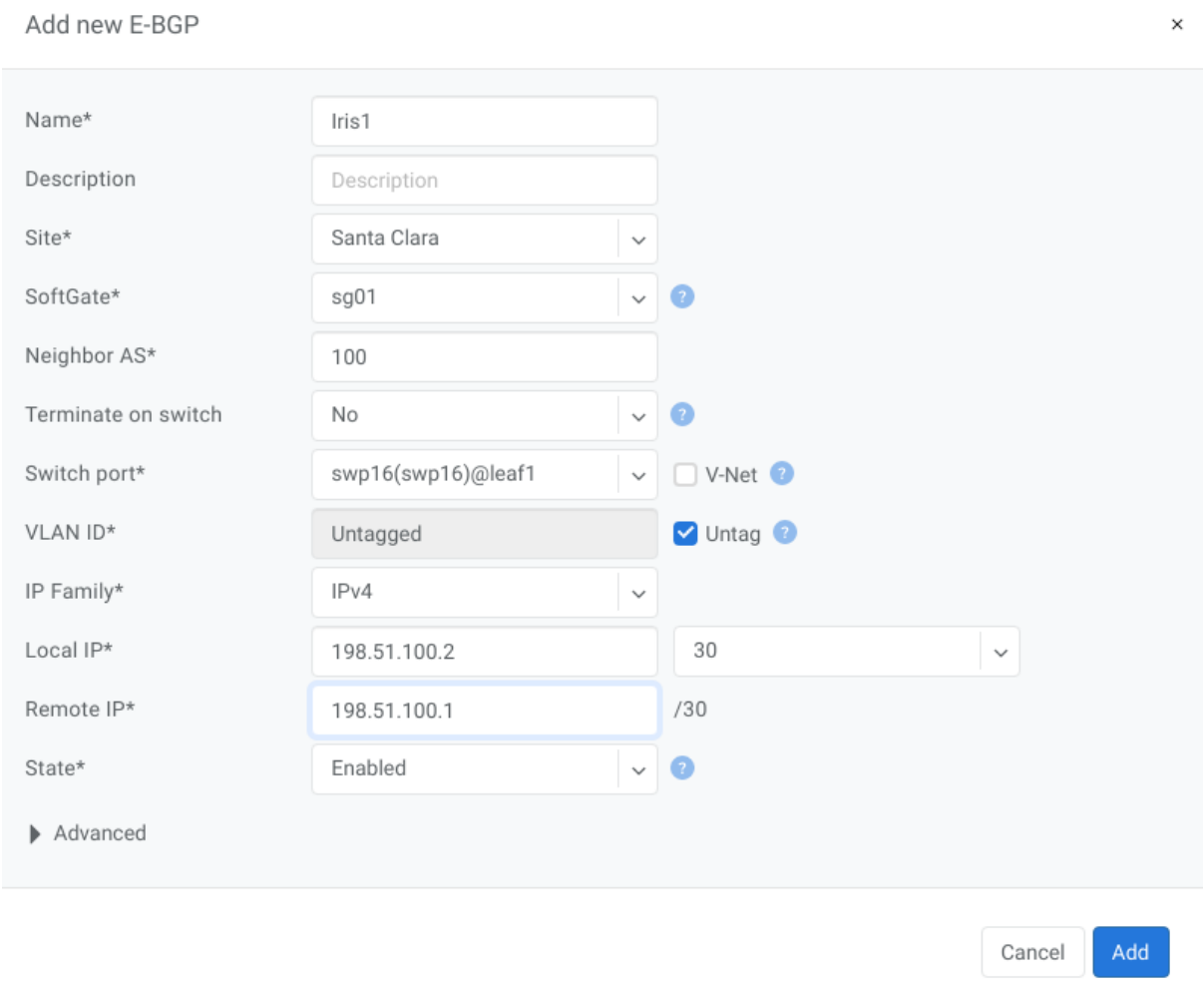
16.1 Adding BGP Peers

1. Navigate to Net→E-BGP in the web UI
2. Click the **Add** button
3. Fill in the fields as described in the table below
4. Click the **Add** button

Table 16.1. BGP Peer Fields

Name	User assigned name of BGP session
Description	Free description
Site	Selects the site (data center) where this BGP session should be terminated on.
Softgate	Only if SoftGate nodes are in use. Define on which node BGP session should be terminated on.
Neighbor AS	Autonomous System number of the remote side. (Local AS is defined at Net→Sites section)
Terminate on switch	Typically used for setups without SoftGate. For connecting with upstream routers. Instructs the system to terminate the BGP session directly on the switch.
Switch port	Switch Port for the physical cable to the BGP neighbor (any port on the fabric). Optionally can bind to a V-NET service. Typically used for peering with IXPs or systems like GGC (Google Global Cache).
VLAN ID	Optionally tag with a VLAN ID (usually untagged)
IP Version	IPv4 or IPv6
Local IP	BGP peering IP address on Netris controlled side
Remote IP	BGP peering IP address on the remote end
State	Administrative state (Enabled/Disabled)
Advanced	Advanced policy settings are described in the next section

Example: Declare a basic BGP neighbor



Example2: Declare BGP neighbor terminated on V-Net. Netris will automatically configure BGP session on the switch closest to the remote IP. .. image:: images/add-bgp-basic-2.png

align center

class with-shadow

Advanced BGP

BGP neighbor declaration can optionally include advanced BGP attributes and BGP route-maps for fine-tuning of BGP policies.

Click Advanced to expand the BGP neighbor add/edit window.

Table 17.1. BGP Peer Fields - Advanced

Neighbor address	IP address of the neighbor when peering with the loopback IP address instead of the interface IP address (aka Multihop).
Update source	When Multihop BGP peering is used it allows the operator to choose one of the loopback IP addresses of the SoftGate node as a BGP speaker source IP address.
BGP password	Password for the BGP session
Allowas-in	Define the number of allowed occurrences of the self AS number in the received BGP NLRI to consider it valid (normally 0).
Default Originate	Originate default route to the current neighbor
Prefix Inbound Max	Drop the BGP session if the number of received prefixes exceeds this max limit. For switch termination maximum allowed is 1000 prefixes and SoftGate termination can handle up to one million prefixes.
Inbound Route-Map	Apply BGP policies described in a route-map for inbound BGP updates
Outbound Route-Map	Apply BGP policies described in a route-map for outbound BGP updates
Local Preference	Set local preference for all inbound routes for the current neighbor
Weight	Set weight for all inbound routes for the current neighbor
Prepend Inbound (times)	How many times to prepend self AS number for inbound routes
Prepend Outbound (times)	How many times to prepend self AS number for outbound routes
Prefix List Inbound	List of IP addresses prefixes to permit or deny inbound
Prefix List Outbound	List of IP addresses prefixes to permit or deny outbound
Send BGP Community	List of BGP communities to send to the current neighbor

17.1 BGP Objects

Under Net→E-BGP objects, you can define various BGP objects referenced from a route-map to declare a dynamic BGP policy.
Supported objects include:

- IPv4 Prefix

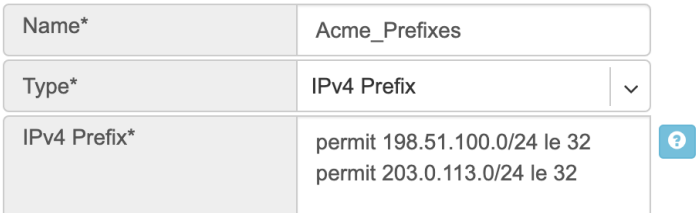
- IPv6 Prefix
- AS-PATH
- Community
- Extended Community
- Large Community

17.1.1 IPv4 Prefix

Rules defined one per line.
Each line in IPv4 prefix list field consists of three parts:

- Action - Possible values are: permit or deny (mandatory).
- IP Prefix - Any valid IPv4 prefix (mandatory).
- Length - Possible values are: le <len>, ge <len> or ge <len> le <len>.

Example: Creating an IPv4 Prefix list.

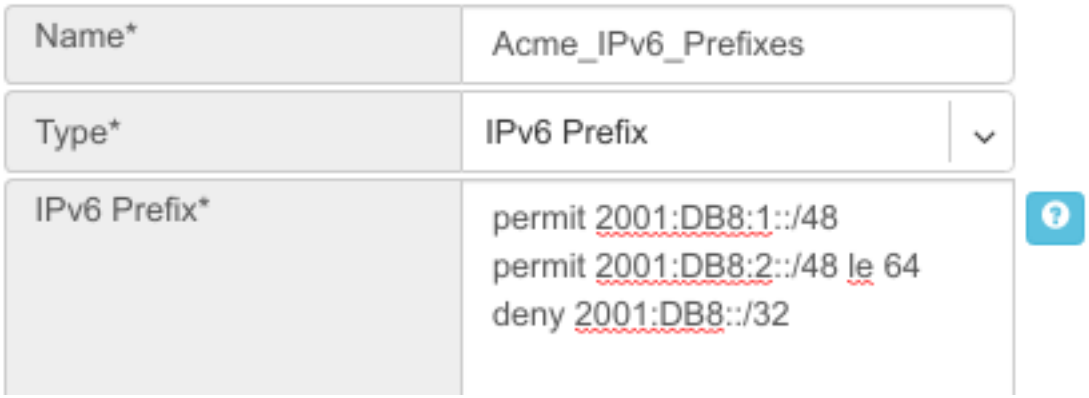


17.1.2 IPv6 Prefix

Rules defined one per line.
Each line in IPv6 prefix list field consists of three parts:

- Action - Possible values are: permit or deny (mandatory).
- IP Prefix - Any valid IPv6 prefix (mandatory).
- Keyword - Possible values are: le <len>, ge <len> or ge <len> le <len>.

Example: Creating an IPv6 Prefix list.



17.1.3 Community

Community field has two parts:

- Action - Possible values: permit or deny (mandatory).

- Community string - format is AA:NN, where AA and NN are any number from 0 to 65535 range or alternatively well known string (local-AS | no-advertise | no-export | internet | additive).

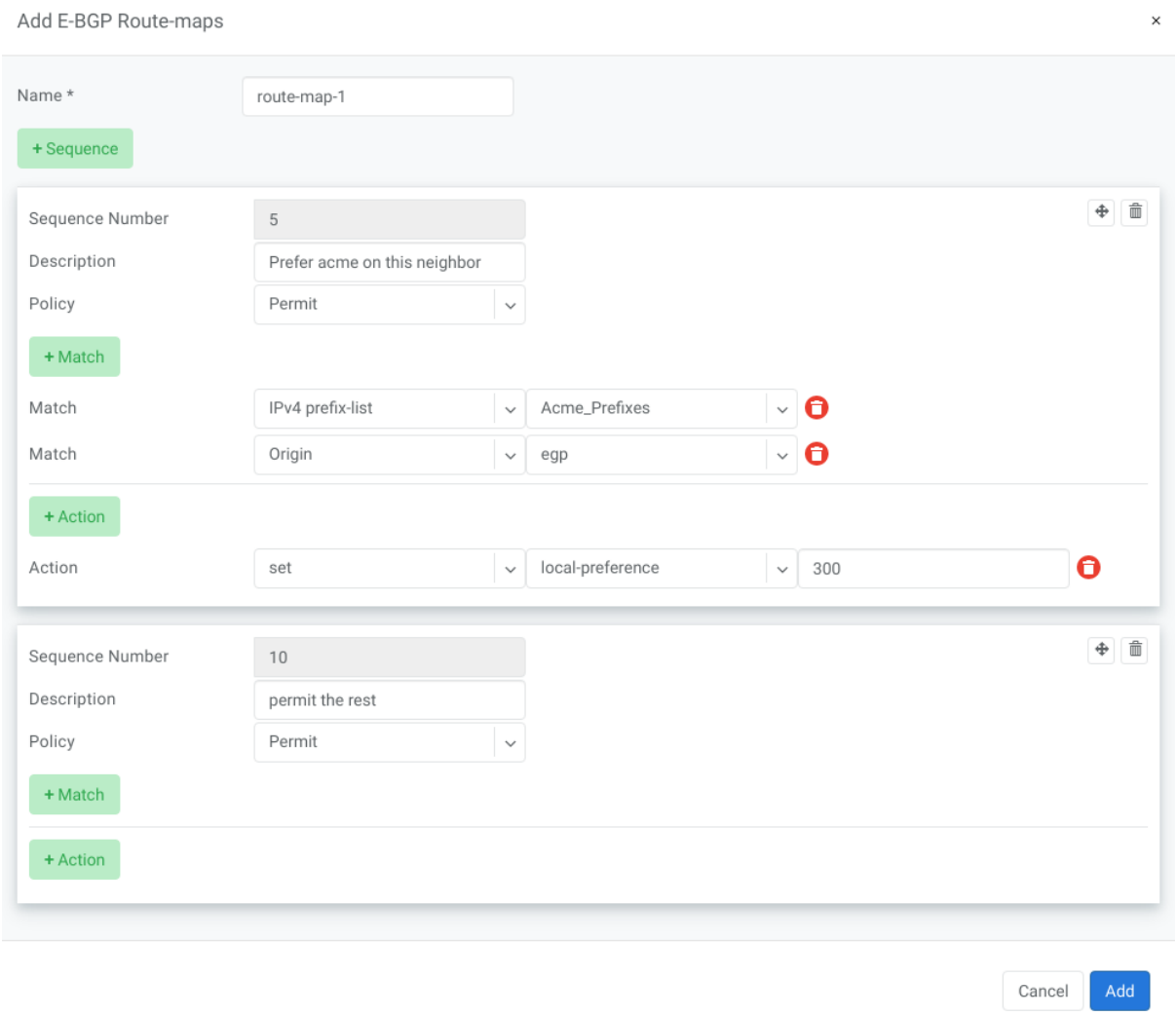
Example: Creating community.

17.2 BGP route-maps

Under the Net→E-BGP Route-maps section, you can define route-map policies, which can be associated with the BGP neighbors inbound or outbound.
Description of route-map fields:

- **Sequence Number** - Automatically assigned a sequence number. Drag and move sequences to organize the order.
- **Description** - Free description.
- **Policy** - Permit or deny the routes which match below all match clauses within the current sequence.
- **Match** - Rules for route matching.
  - **Type** - Type of the object to match: AS-Path, Community, Extended Community, Large Community, IPv4 prefix-list, IPv4 next-hop, Route Source, IPv6 prefix-list. IPv6 next-hop, local-preference, MED, Origin, Route Tag.
  - **Object** - Select an object from the list.
- **Action** - Action when all match clauses are met.
  - **Action type** - Define whether to manipulate a particular BGP attribute or go to another sequence.
  - **Attribute** - The attribute to be manipulated.
  - **Value** - New attribute value.

Example: route-map


Add E-BGP Route-maps
Name *
route-map-1
+ Sequence
Sequence Number
5
Description
Prefer acme on this neighbor
Policy
Permit
+ Match
Match
IPv4 prefix-list
Acme_Prefixes
Match
Origin
egp
+ Action
Action
set
local-preference
300
Sequence Number
10
Description
permit the rest
Policy
Permit
+ Match
+ Action
Cancel
Add

Static Routing

Located under Net→Routes is a method for describing static routing policies that Netris will dynamically inject on switches and/or SoftGate where appropriate. We recommend using the Routes only if BGP is not supported by the remote end.

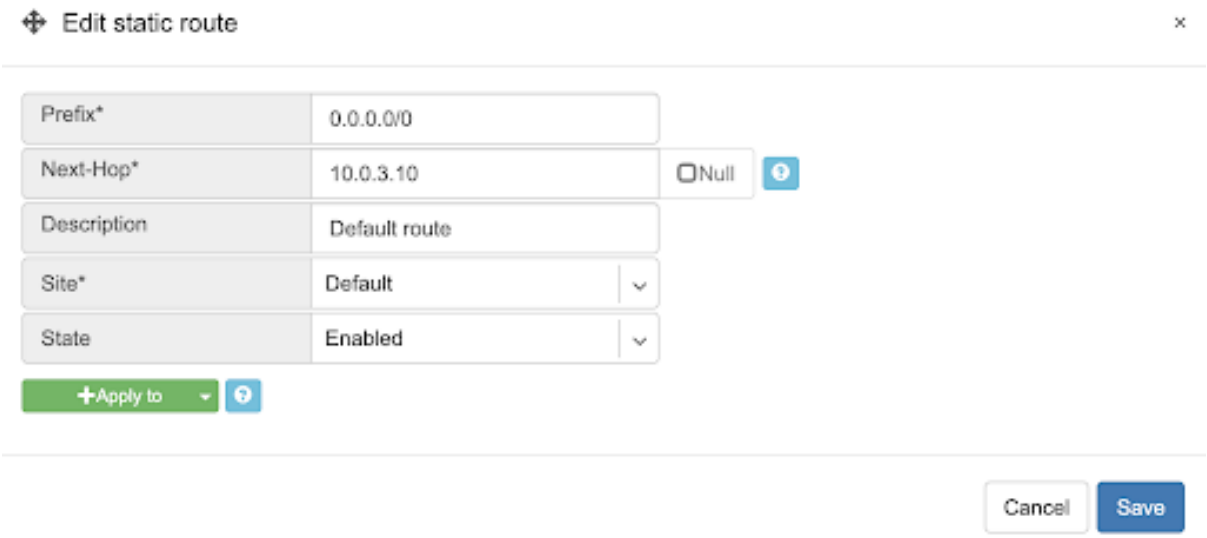
Typical use cases for static routing:

- To connect the switch fabric to an ISP or upstream router in a situation where BGP and dual-homing are not supported.
- Temporary interconnection with the old network for a migration.
- Routing a subnet behind a VM hypervisor machine for an internal VM network.
- Specifically routing traffic destined to a particular prefix through an out-of-band management network.

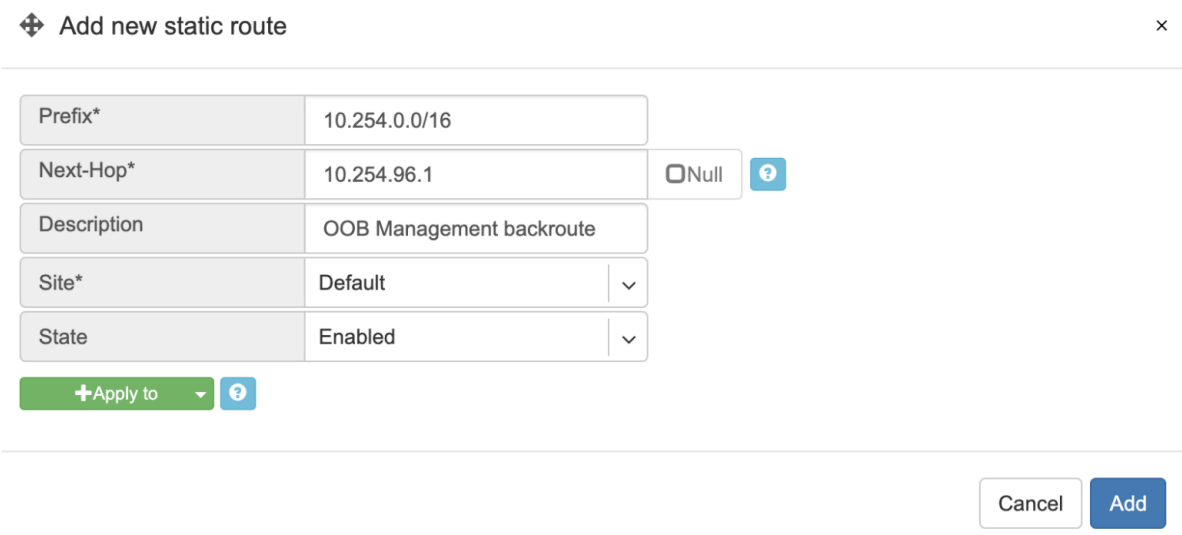
Add new static route fields description:

- **Prefix** - Route destination to match.
- **Next-Hop** - Traffic destined to the Prefix will be routed towards the Next-Hop. Note that static routes will be injected only on units that have the Next-Hop as a connected network.
- **Description** - Free description.
- **Site** - Site where Route belongs.
- **State** - Administrative (enable/disable) state of the Route.
- **Apply to** - Limit the scope to particular units. It's typically used for Null routes.

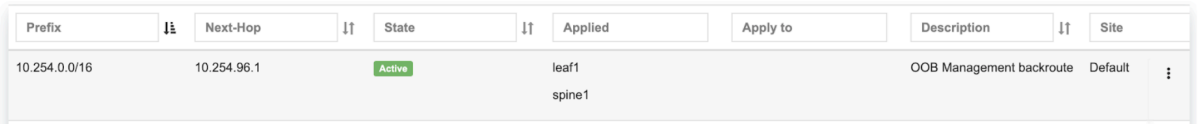
Example: Default route pointing to a Next-Hop that belongs to one of V-NETs.



Example: Adding a back route to 10.254.0.0/16 through an out-of-band management network.



Screenshot: This Shows that my back route is actually applied on leaf1 and spine1.



NAT

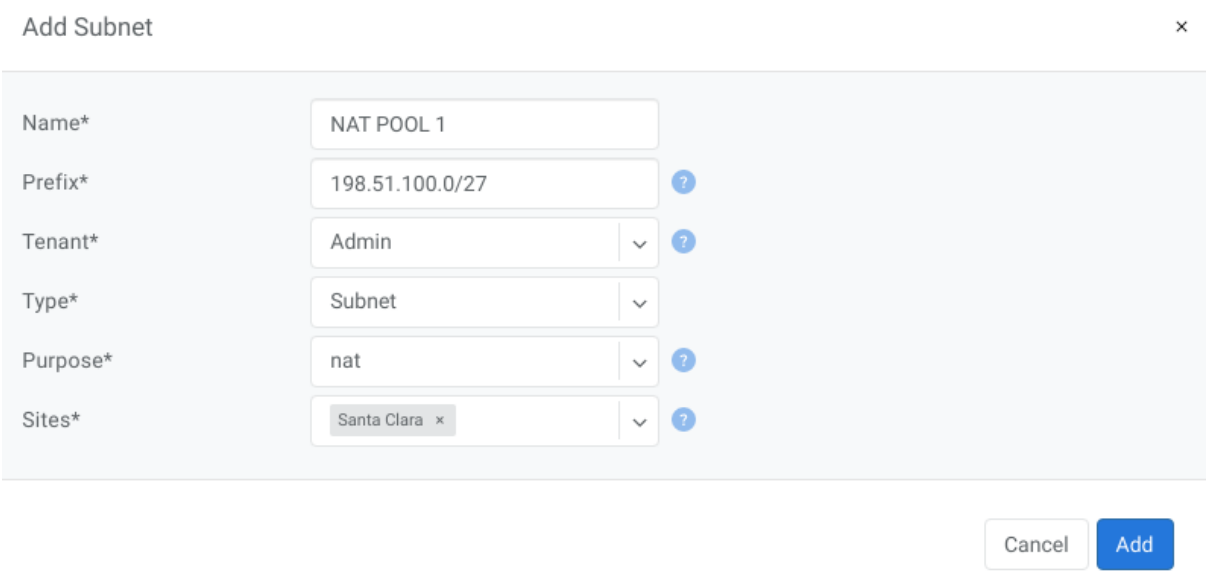
Netris SoftGate nodes are required to support NAT (Network Address Translation).

19.1 Enabling NAT

To enable NAT for a given site, you first need to create a subnet with NAT purpose in the IPAM section. NAT IP addresses can be used for SNAT or DNAT as a global IP address (the public IP visible on the Internet). NAT IP pools are IP address ranges that SNAT can use as a rolling global IP (for a larger scale, similar to carrier-grade SNAT). SNAT is always overloading the ports, so many local hosts can share one or just a few public IP addresses. You can add as many NAT IP addresses and NAT pools as you need.

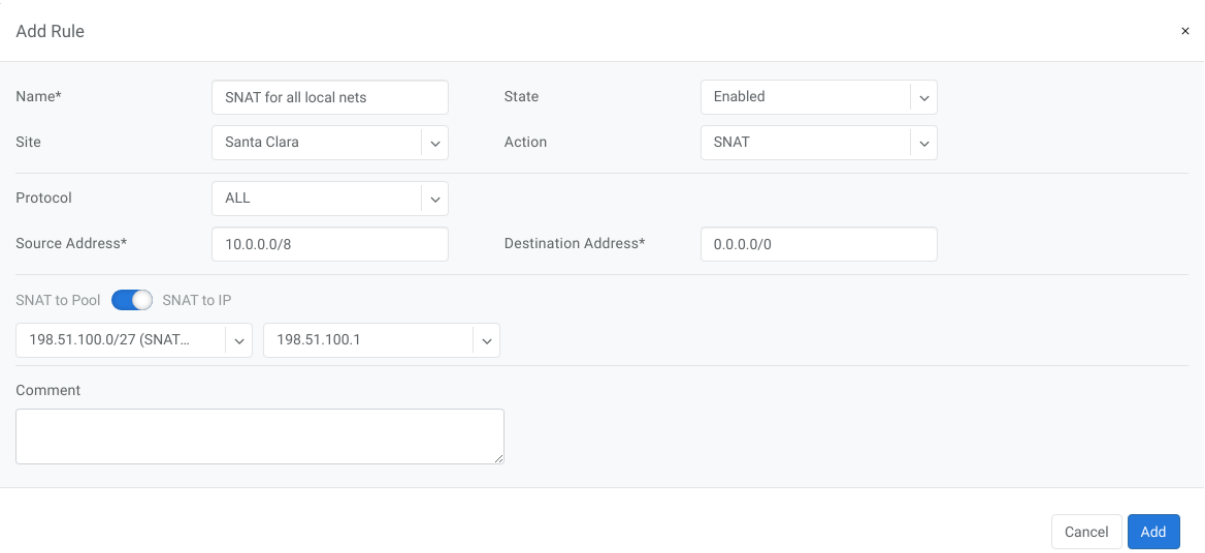
1. Allocate a public IP subnet for NAT under Net→IPAM.

Example: Adding an IP allocation under Net→Subnets.



1. Attach NAT IP addresses and/or NAT IP Pools to just one SoftGate node. Other SoftGate Nodes on the same site will automatically add the same NAT IP/Pool resources for proper consistency and high availability.

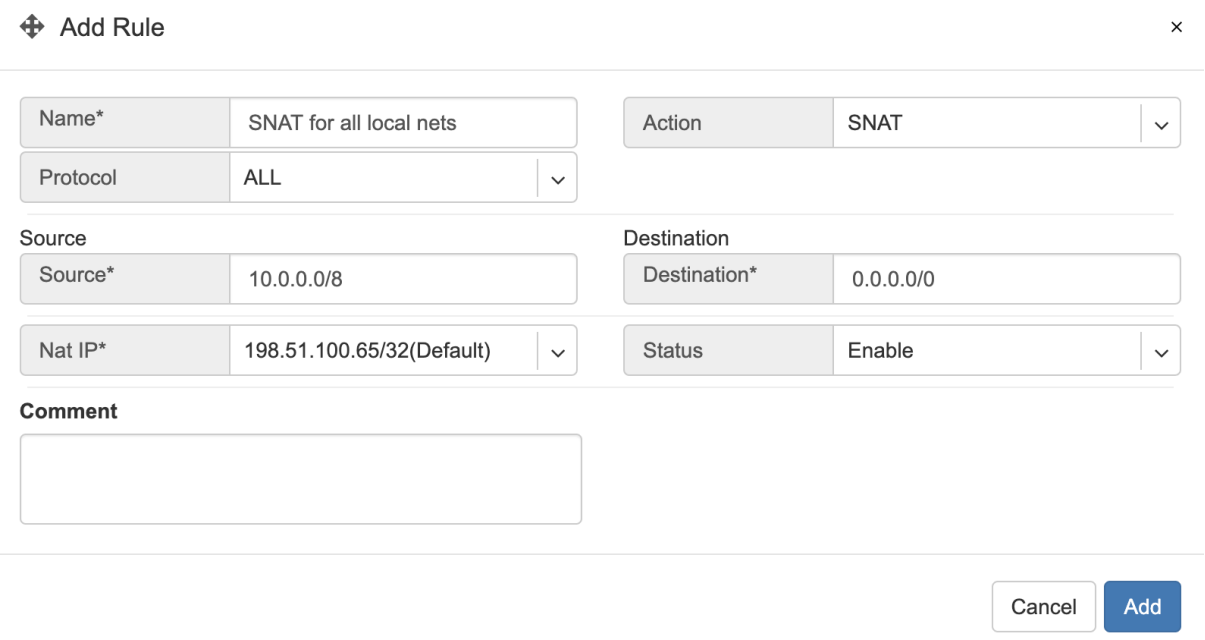
Example: Adding NAT IP addresses and NAT IP Address Pools to a SoftGate node.



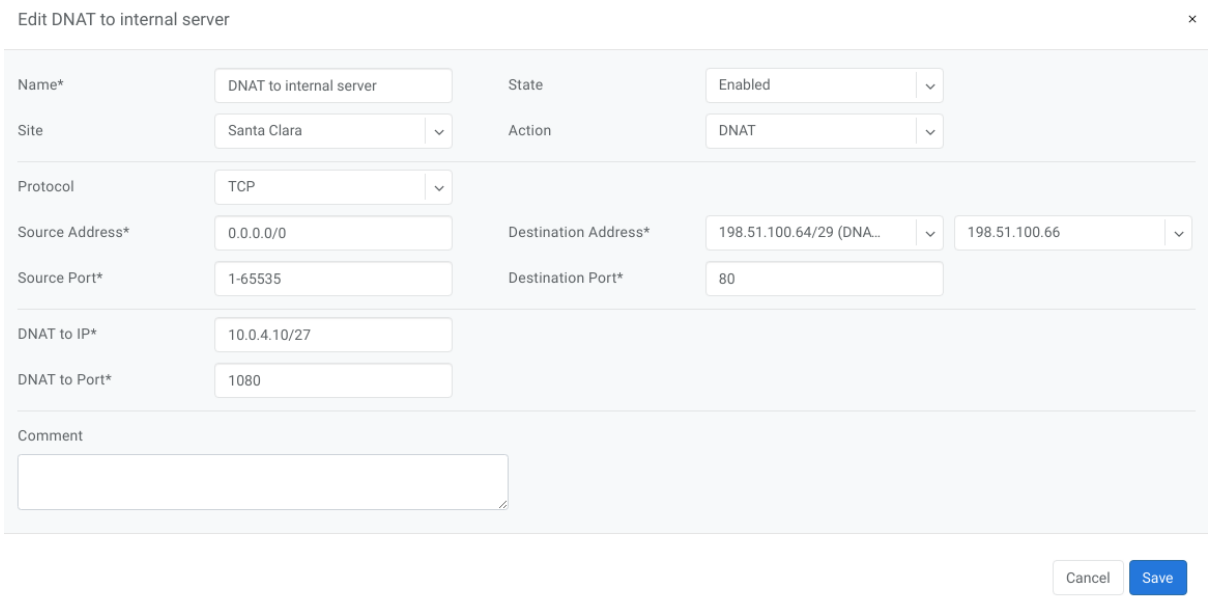
19.2 Defining NAT rules

NAT rules are defined under Net→NAT.

Example: SNAT all hosts on 10.0.0.0/8 to the Internet using 198.51.100.65 as a global IP.



Example: Port forwarding. DNAT the traffic destined to 198.51.100.66:80 to be forwarded to the host 10.0.4.10 on port tcp/1080.


Edit DNAT to internal server
Name*
DNAT to internal server
State
Enabled
Site
Santa Clara
Action
DNAT
Protocol
TCP
Source Address*
0.0.0.0/0
Destination Address*
198.51.100.64/29 (DNA...
198.51.100.66
Source Port*
1-65535
Destination Port*
80
DNAT to IP*
10.0.4.10/27
DNAT to Port*
1080
Comment
Cancel
Save

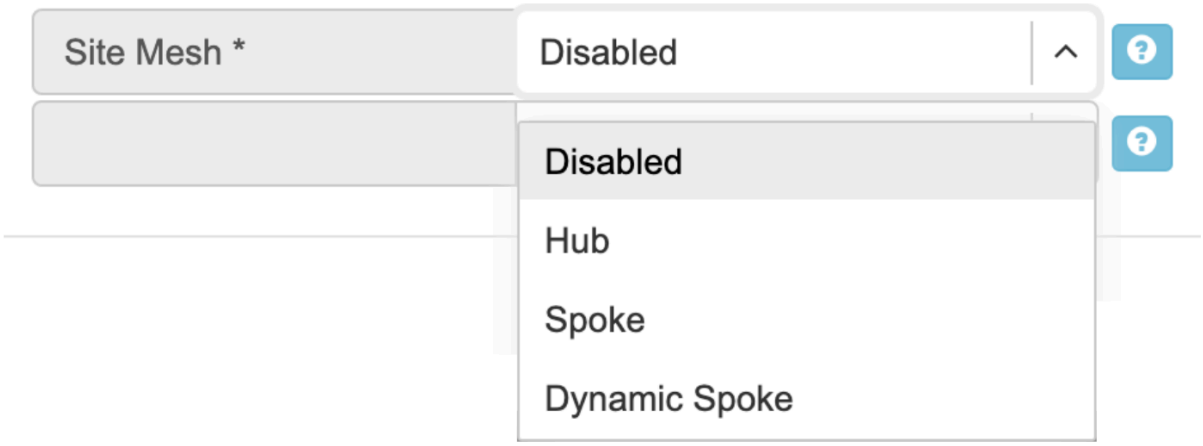
SiteMesh

SiteMesh is a Netris service for site-to-site interconnects over the public Internet. SiteMesh automatically generates configuration for WireGuard to create encrypted tunnels between participating sites and automatically generates a configuration for FRR to run dynamic routing. Hence, sites learn how to reach each other over the mesh WireGuard tunnels. The SiteMesh feature requires a SoftGate node at each participating site.

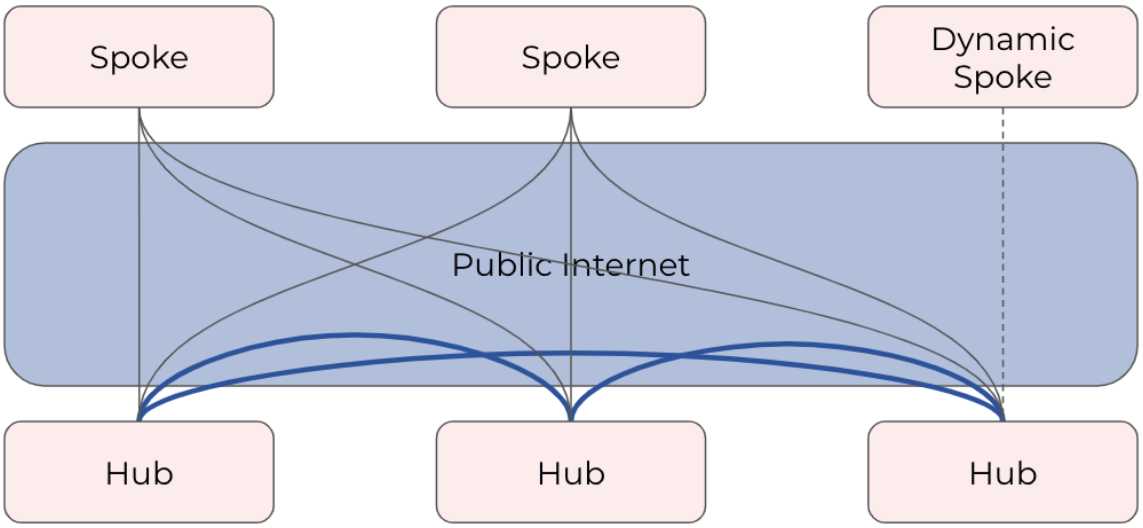
Edit Net->Sites, do declare what sites should form a SiteMesh. See SiteMesh types described below.

- **Disabled** - Do not participate in SiteMesh.
- **Hub** - Hub sites form full-mesh tunnels with all other sites (Hub and non-Hub) and can carry transit traffic for non-Hub sites. (usually major data center sites)
- **Spoke** - Spoke sites form tunnels with all Hub sites. Spoke to Spoke traffic will transit a Hub site. (small data center sites or major office sites)
- **Dynamic Spoke** - Dynamic Spoke is like Spoke, but it will maintain a tunnel only with one Hub site, based on dynamic connectivity measurements underneath and mathematical modeling. (small office sites)

Screenshot: Site Mesh parameter editing a Site under Net→Sites.

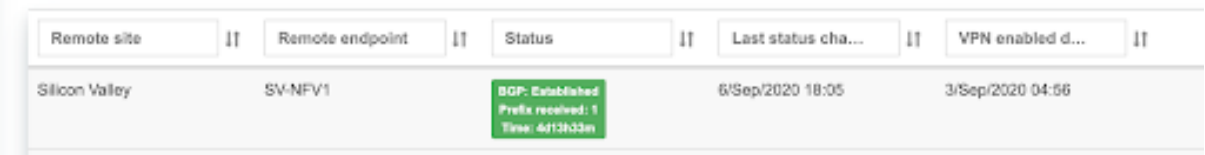


You only need to define your site-to-site VPN architecture policy by selecting SiteMesh mode for every site. Netris will generate the WireGuard tunnels (using randomly generated keys, and generate FRR rules to get the dynamic routing to converge.



Check the Net→Site Mesh section for the listing of tunnel statuses.

Screenshot: Listing of SiteMesh tunnels and BGP statuses (Net→Site Mesh)



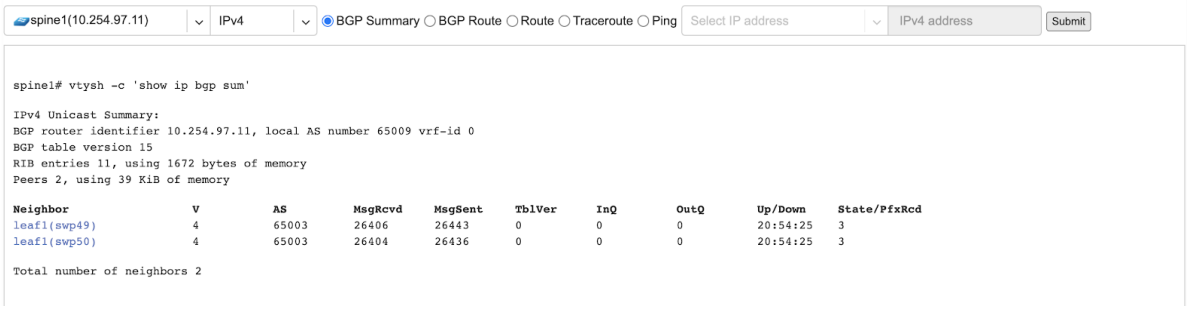
Looking Glass

The Looking Glass Is a GUI-based tool for looking up routing information from a switch or SoftGate perspective. You can access the Looking Glass either from Topology, individually for every device (right click on device → details → Looking Glass), or by navigating to Net→Looking Glass then selecting the device from the top-left dropdown menu.

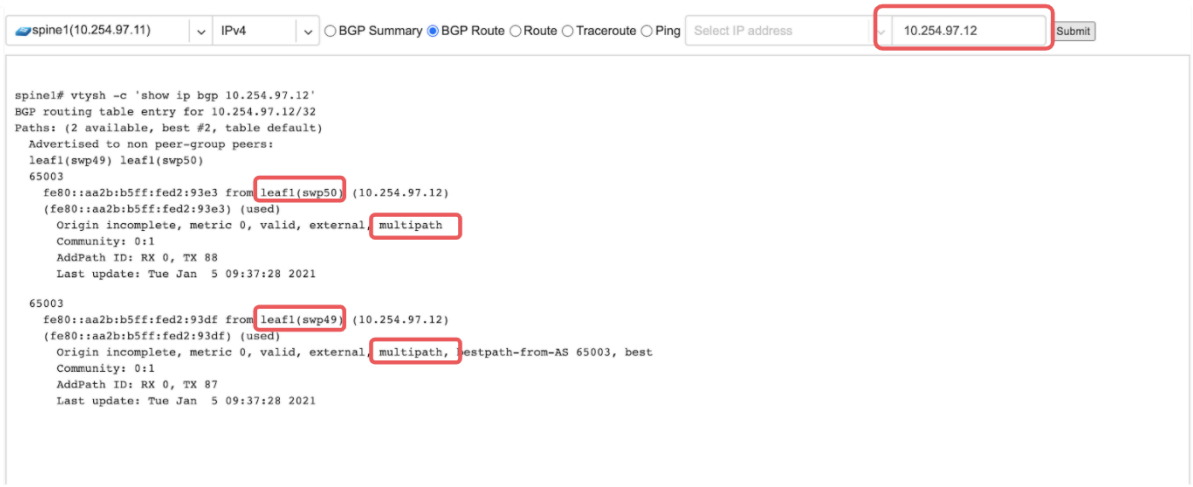
Looking Glass controls described for IPv4/IPv6 protocol families.

- **BGP Summary** - Shows the summary of BGP adjacencies with neighbors, interface names, prefixes received. You can click on the neighbor name then query for the list of advertised/received prefixes.
- **BGP Route** - Lookup the BGP table (RIB) for the given address.
- **Route** - Lookup switch routing table for the given address.
- **Traceroute** - Conduct a traceroute from the selected device towards the given destination, optionally allowing to determine the source IP address.
- **Ping** - Execute a ping on the selected device towards the given destination, optionally allowing to select the source IP address.

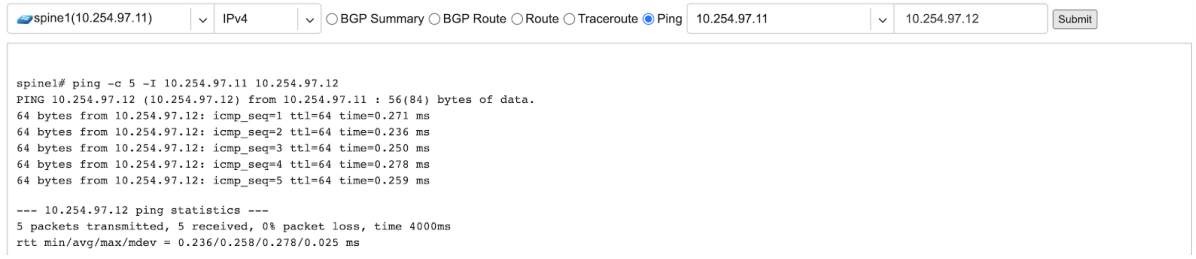
Example: Spine1: listing BGP neighbors and number of received prefixes.



Example: BGP Route - looking up my leaf1 switch's loopback address from spine1's perspective. Spine1 is load balancing between two available paths.



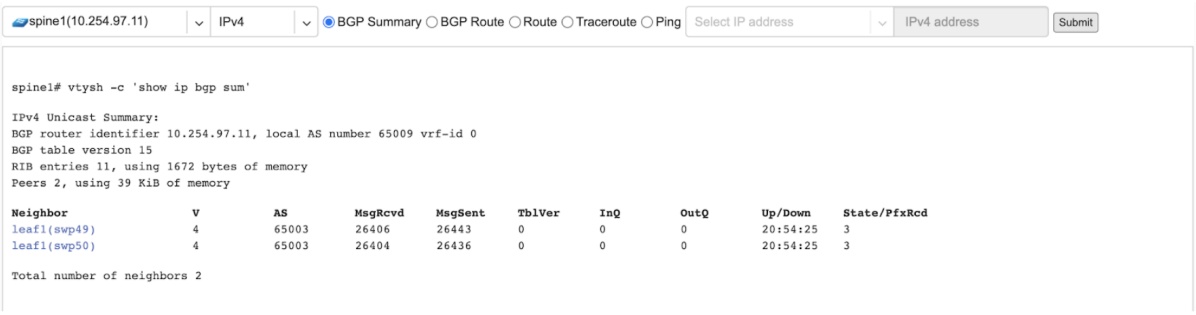
Example: Ping.



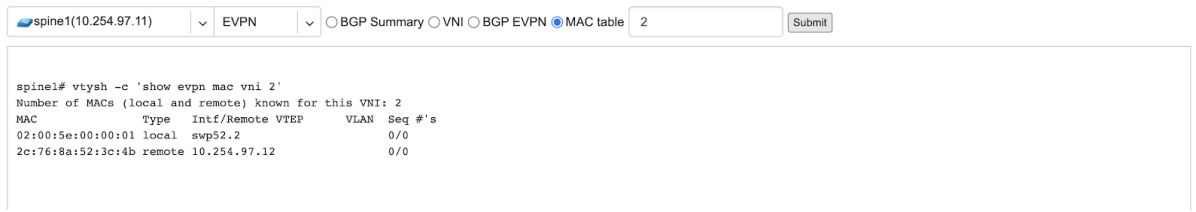
Looking Glass controls described for the EVPN family.

- **BGP Summary** - Show brief summary of BGP adjacencies with neighbors, interface names, and EVPN prefixes received.
- **VNI** - List VNIs learned.
- **BGP EVPN** - List detailed EVPN routing information optionally for the given route distinguisher.
- **MAC table** - List MAC address table for the given VNI.

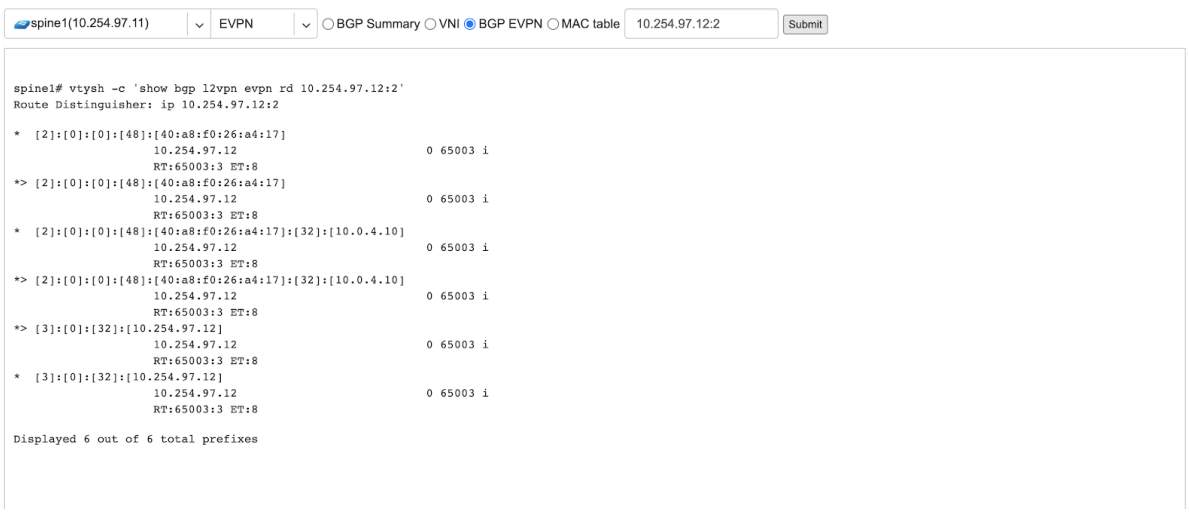
Example: Listing of adjacent BGP neighbors and number of EVPN prefixes received.



Example: Listing MAC addresses on VNI 2.



Example: EVPN routing information listing for a specified route distinguisher.



Chapter 22

V-Net

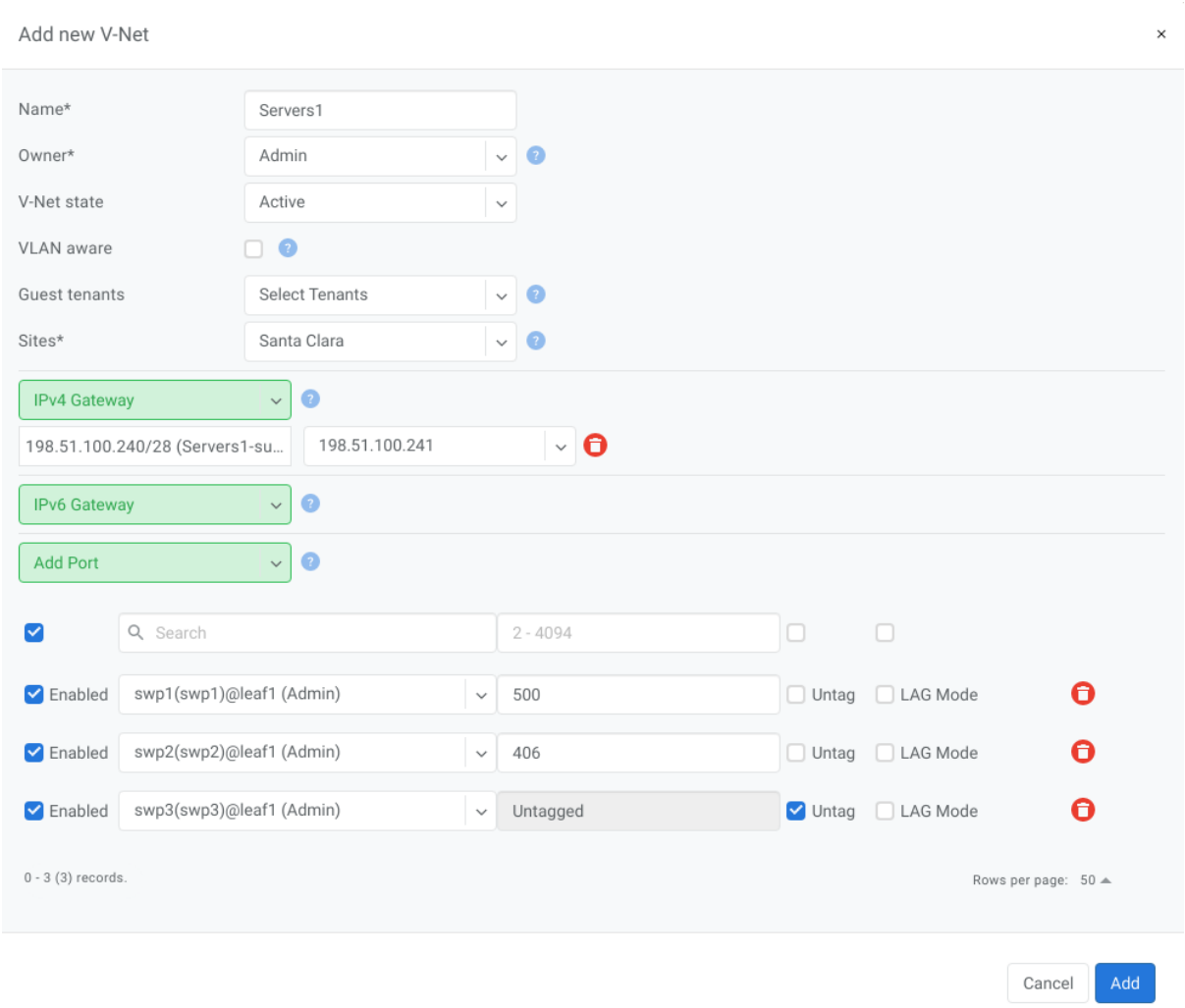
V-Net is a virtual networking service that provide a Layer-2 (unrouted) or Layer-3 (routed) virtual network segments on switch ports anywhere on the switch fabric. V-NETs can be created and managed by a single tenant (single team) or they can be created and managed collaboratively by multiple tenants (different teams inside and/or outside the organization). Netris automatically configures a VXLAN with an EVPN control plane over an unnumbered BGP Layer-3 underlay network and organize the high availability for the default gateway behind the scenes.

22.1 V-Net Fields

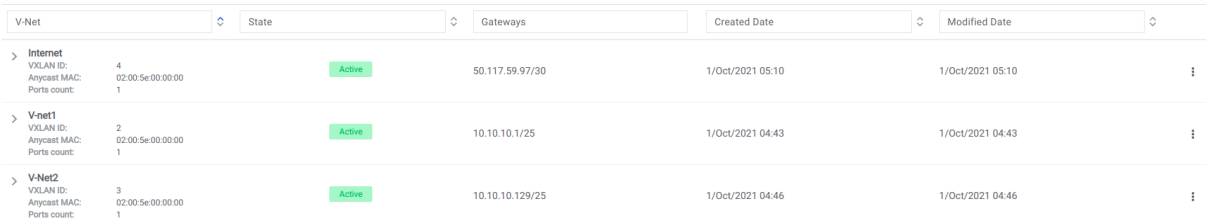
- **Name** - Unique name for the V-Net
- **Owner** - Tenant, who can make any changes to current V-Net
- **V-Net state** - Active/Disable state for entire V-Net
- **VLAN aware** - Enable VLAN aware bridge, use only in rare cases, if otherwise is not possible
- **Guest tenants** - List of tenants allowed to add/edit/remove ports to the V-Net but not manage other parameters
- **Sites** - Ports from these sites will be allowed to participate in the V-Net. (Multi-site circuits would require backbone connectivity between sites).
- **IPv4 Gateway** - IPv4 address to be used as a default gateway in this V-Net. Should be configured under Net→IPAM as an assignment, assigned to the owner tenant, and available in the site where V-Net is intended to span.
- **IPv6 Gateway** - IPv6 address to be used as a default gateway in this V-Net. Should be configured under Net→IPAM as an assignment, assigned to the owner tenant, and available in the site or sites where V-Net is intended to span.
- **Port** - Physical Switch Port anywhere on the network. Switch Port should be assigned to the owner or guest tenant under Net→Switch Ports.
    - **Enabled** - Enable or disable individual Switch Port under current V-Net
    - **Port Name** - Switch Port format: <alias>(swp<number>)@<switch name>
    - **VLAN ID / Untag** - Specify a VLAN ID for tagging traffic on a per-port basis or set Untag not to use tagging on a particular port. VLAN tags are only significant on each port's ingress/egress unless VLAN aware mode is used.
    - **LAG Mode** - Allows for active-standby dual-homing, assuming LAG configuration on the remote end. Active/active dual homing will be enabled in future releases (dependence on SVI support by NOSes).

Tip Many switches can't autodetect old 1Gbps ports. If attaching hosts with 1Gbps ports to 10Gpbs

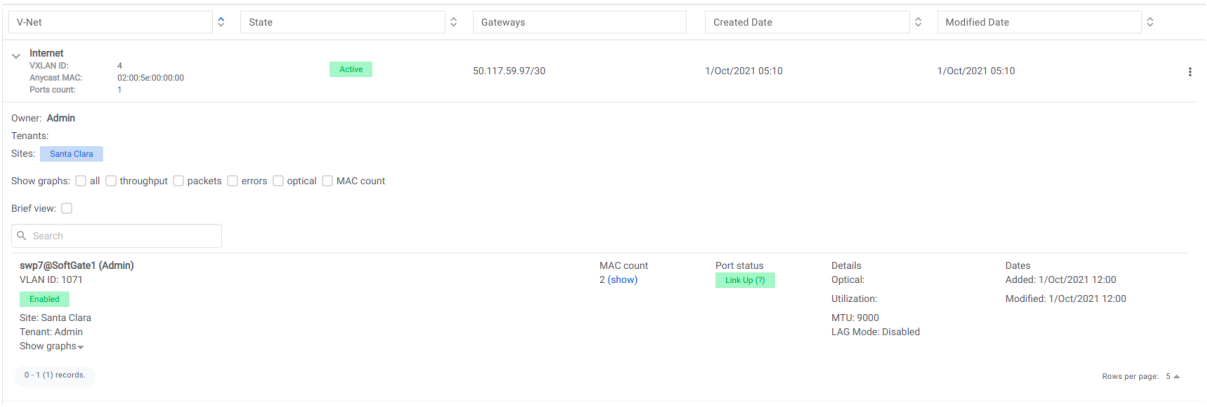
switch ports, you'll need to change the speed for a given Switch Port from Auto(default) to 1Gbps. You can edit a port in Net→Switch Ports individually or in bulk.



Example: Adding a new V-Net.



Example: Listing of V-Nets.



Example: Expanded view of a V-Net listing.

L3 Load Balancer (Anycast LB)

L3 (Anycast) load balancer leverages ECMP load balancing and hashing capability of spine and leaf switches to deliver line-rate server load balancing with health checks.

ROH servers, besides advertising their unicast (unique) loopback IP address, need to configure and advertise an additional anycast (the same IP) IP address. Unicast IP address is used for connecting to each individual server.

End-user traffic should be destined to the anycast IP address. The switch fabric uses ECMP to load balance the traffic towards every server, and will hash sessions based on IP/Protocol/Port such that TCP sessions will exist between the given end-user and server pair for the lifetime of the session. Optional health checks are used to identify application failures and reroute traffic in the case of a server outage.

23.1 Creating an L3 Load Balancer

To configure L3 (Anycast) load balancing:

1. Navigate to **Services→Instances (ROH)** and locate an existing ROH instance
2. Click the ellipses and then the **Edit** button
3. Select an extra IPv4 address from the select box at the bottom, and check the **Anycast** option.
4. This will create a service under Services→Load Balancer and permit using the Anycast IP address in multiple ROH instances.

L4 Load Balancer (L4LB)

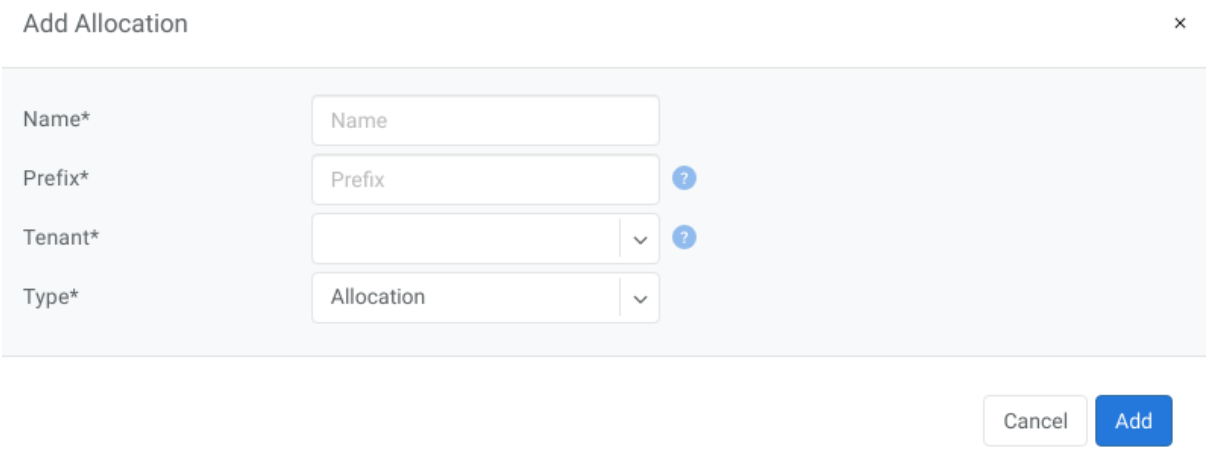
Netris L4 Load Balancer (L4LB) leverages SoftGate(Linux router) nodes to provide Layer-4 load balancing services, including on-demand cloud load balancing with native integration with Kubernetes.

24.1 Enabling L4LB service

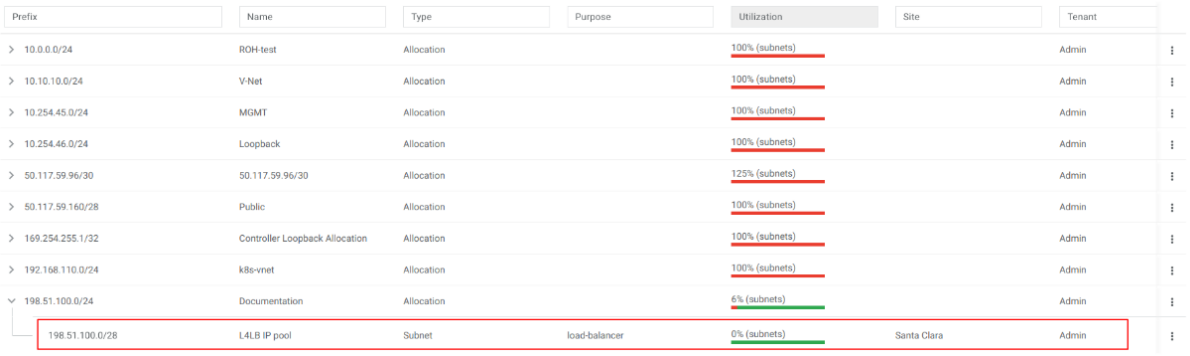
L4 Load Balancer service requires at least one SoftGate node to be available in a given Site, as well as at least one IP address assignment (purpose=load balancer).

The IP address pool for L4LB can be defined in the Net→IPAM section by adding an Allocation and setting the purpose field to 'load-balancer'. You can define multiple IP pools for L4LB at any given site. See the below example.

Example: Adding a load-balancer IP pool assignment.



Screenshot: Listing of Net→IPAM after adding a load-balancer assignment



24.2 Consuming L4LB service

This guide describes how to request an L4 Load Balancer using GUI. For Kubernetes integration, check the Kubenet section.

Click +add under Services→L4 Load Balancer to request an L4LB service.

Add new L4 Load Balancer fields are described below:

General fields

- **Name** - Unique name.
- **Protocol** - TCP or UDP.
- **Tenant** - Requestor Tenant should have access to the backend IP space.
- **Site** - Site where L4LB service is being requested for. Backends should belong on this site.
- **State** - Administrative state.

Frontend

- **Address** - Frontend IP address to be exposed for this L4LB service. “Assign automatically” will provide the next available IP address from the defined load-balancer pool. Alternatively, users can select manually from the list of available addresses.
- **Port** - TCP or UDP port to be exposed.

Health-check

- **Type** - Probe backends on service availability.
  - **None** - load balance unconditionally.
  - **TCP** - probe backend service availability through TCP connect checks.
  - **HTTP** - probe backend service availability through HTTP GET checks.
- **Timeout(ms)** - Probe timeout in milliseconds.
- **Request path** - HTTP request path.

Backend

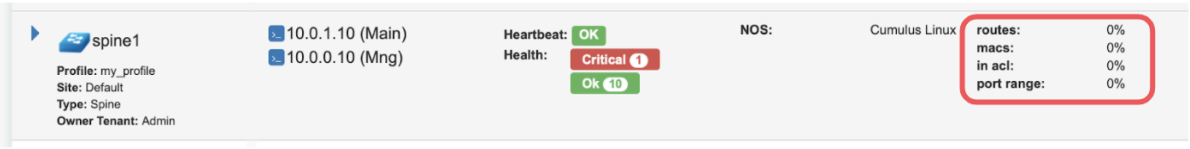
- **+Add** - add a backend host.
- **Address** - IP address of the backend host.
- **Port** - Service port on the backend host.
- **Enabled** - Administrative state of particular backend.

Access Control Lists (ACL)

Netris supports ACLs for switch network access control. (ACL and ACL2.0) ACL is for defining network access lists in a source IP: Port, destination IP: Port format. ACL2.0 is an object-oriented service way of describing network access.

Both ACL and ACL2.0 services support tenant/RBAC based approval workflows. Access control lists execute in switch hardware providing line-rate performance for security enforcement. It's important to keep in mind that the number of ACLs is limited to the limited size of TCAM of network switches.

Screenshot: TCAM utilization can be seen under Net→Inventory

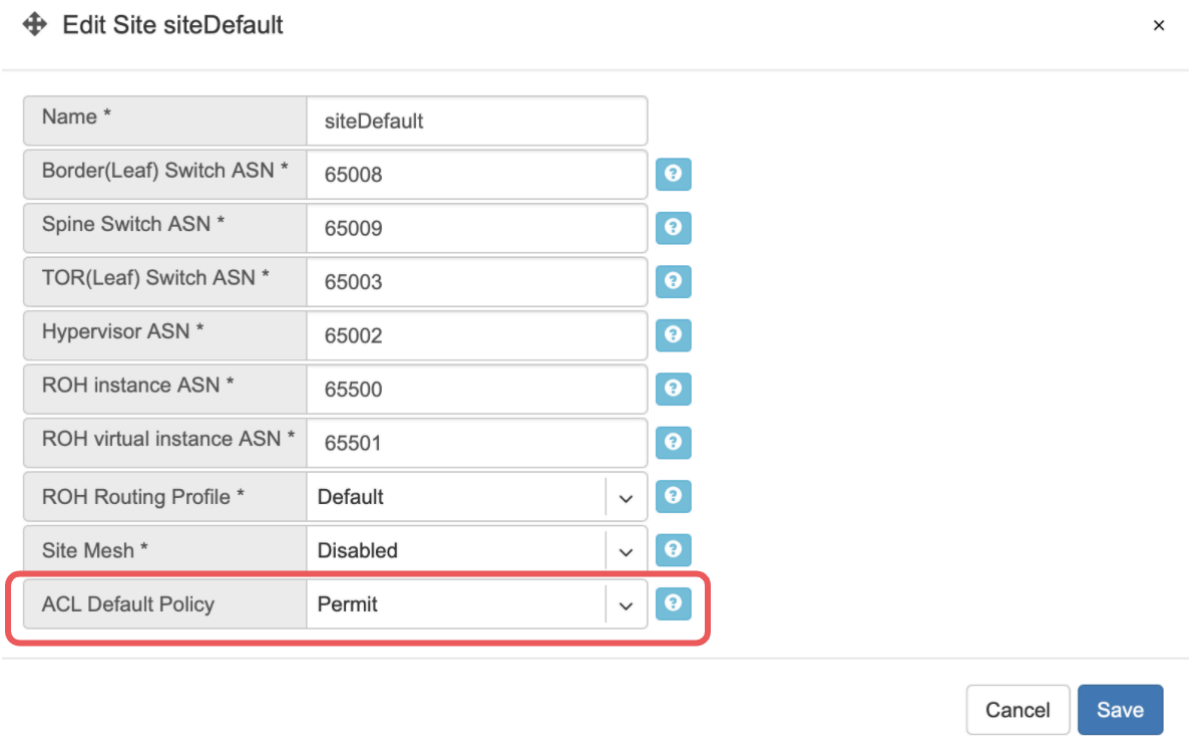


Netris is applying several optimization algorithms to minimize the usage of TCAM while achieving the user-defined requirements.

25.1 ACL Default Policy

The ACL default policy is to permit all hosts to communicate with each other. You can change the default policy on a per Site basis by editing the Site features under Net→Sites. Once the "ACL Default Policy" is changed to "Deny," the given site will start dropping any traffic unless specific communication is permitted through ACL or ACL2.0 rules.

Example: Changing "ACL Default Policy" for the site "siteDefault".



25.2 ACL Rules

ACL rules can be created, listed, edited, approved under Services→ACL.

Description of ACL fields. General

- **Name** - Unique name for the ACL entry.
- **Protocol** - IP protocol to match.
  - All - Any IP protocols.
  - IP - Specific IP protocol number.
  - TCP - TCP.
  - UDP - UDP.
  - ICMP ALL - Any IPv4 ICMP protocol.
  - ICMP Custom - Custom IPv4 ICMP code.
  - ICMPv6 ALL - Any IPv6 ICMP protocol.
  - ICMPv6 Custom - Custom IPv6 ICMP code.
- **Active Until** - Disable this rule at the defined date/time.
- **Action** - Permit or Deny forwarding of matched packets.
- **Established/Reverse** - For TCP, also match reverse packets except with TCP SYN flag. For non-TCP, also generate a reverse rule with swapped source/destination.

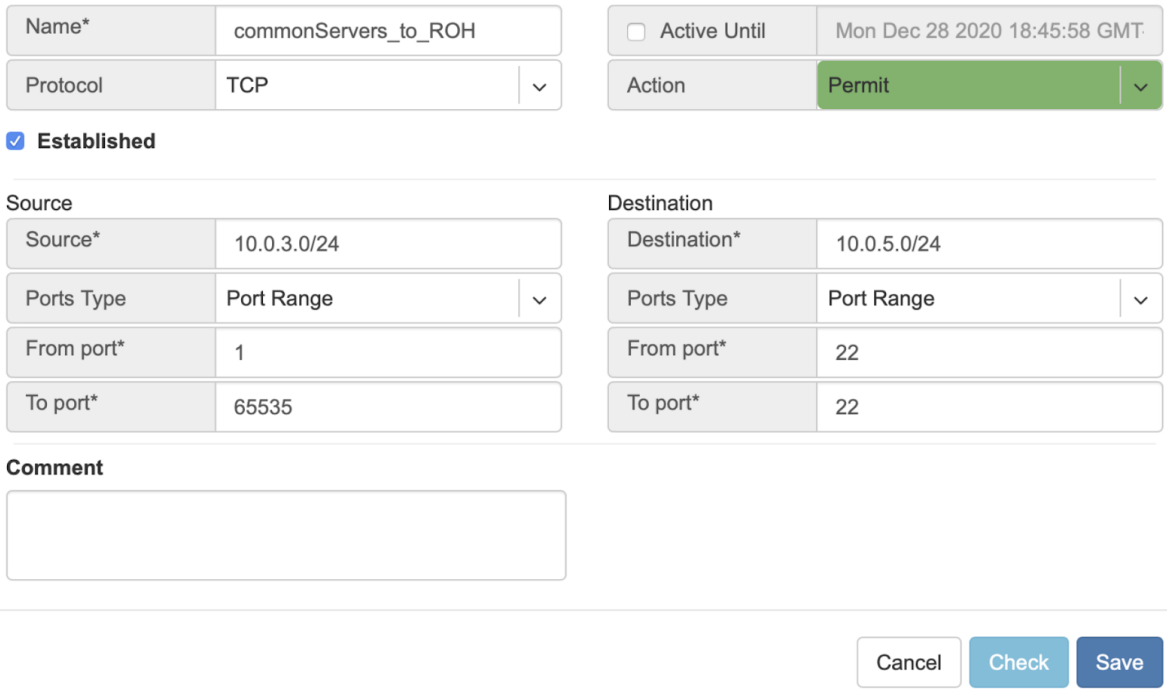
Source/Destination - Source and destination addresses and ports to match.

- **Source** IPv4/IPv6 - IPv4/IPv6 address.
- **Ports Type**
  - Port Range - Match on the port or a port range defined in this window.

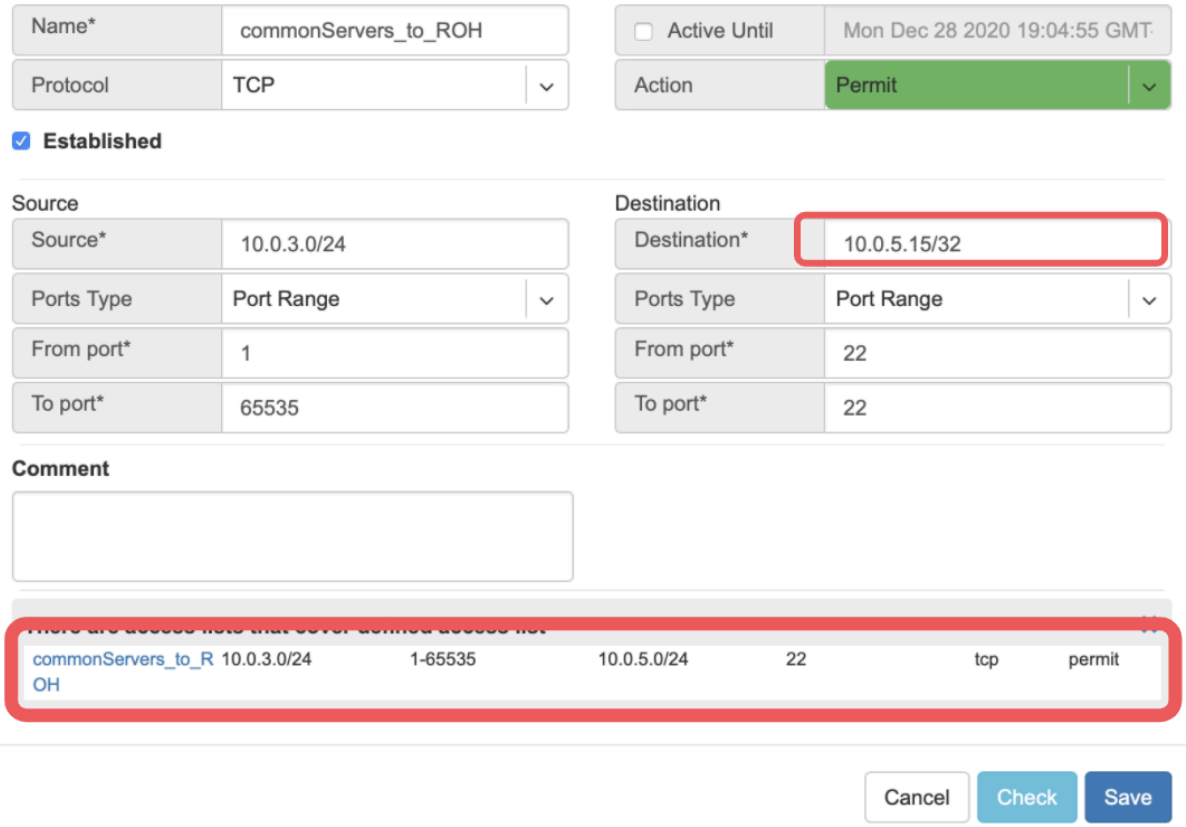
  - Port Group - Match on a group of ports defined under Services→ ACL Port Group.

- **From Port** - Port range starting from.
- **To Port** - Port range ending with.
- **Comment** - Descriptive comment, commonly used for approval workflows.
- **Check button** - Check if Another ACL on the system already permits the described network access.

Example: Permit hosts in 10.0.3.0/24 to access hosts in 10.0.5.0/24 by SSH, also permit the return traffic (Established).



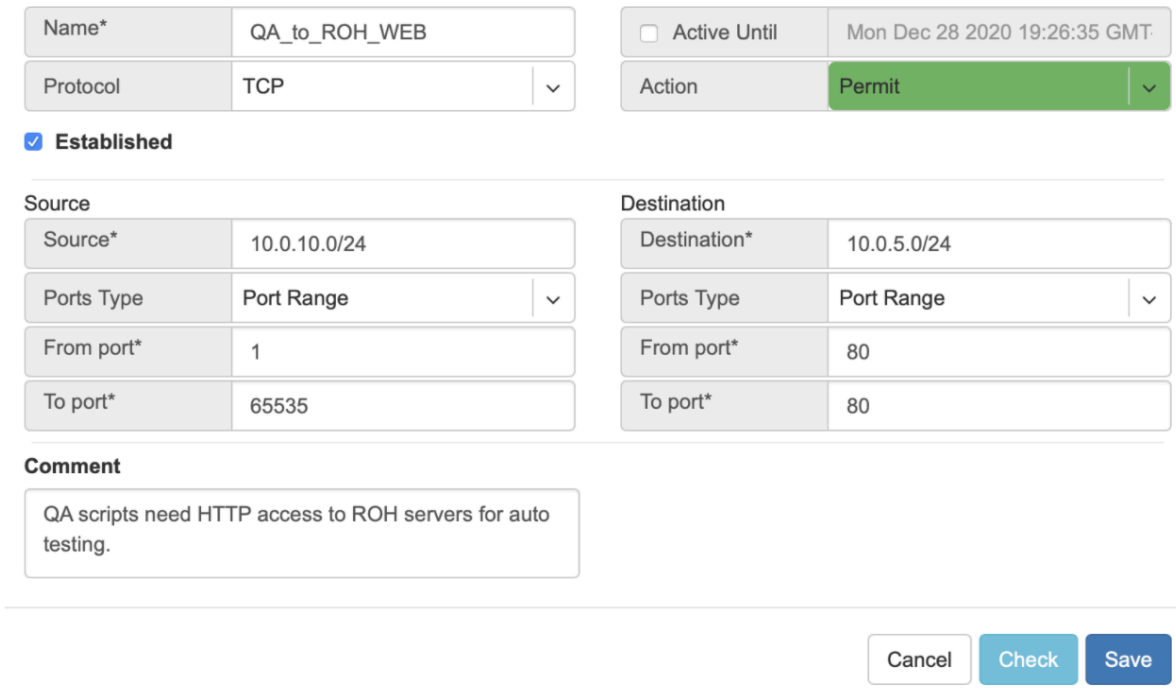
Example: “Check” shows that requested access is already provided by a broader ACL rule.



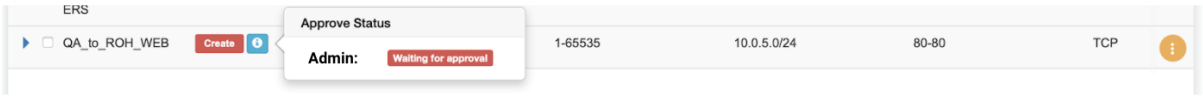
25.3 ACL Approval Workflow

When one tenant (one team) needs to get network access to resources under the responsibility of another tenant (another team), an ACL can be created but will activate only after approval of the tenant responsible for the destination address resources. See the below example.

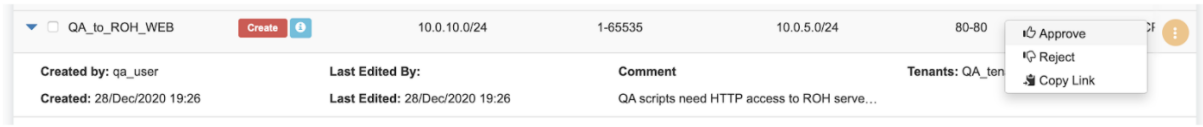
Example: User representing QA_tenant is creating an ACL where source belongs to QA_tenant, but destination belongs to the Admin tenant.



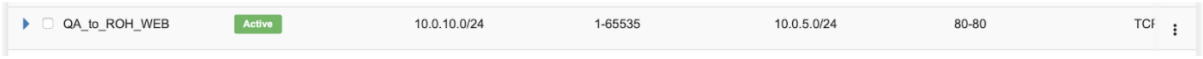
Screenshot: ACL stays in "waiting for approval" state until approved.



Screenshot: Users of tenant Admin, receive a notification in the GUI, and optionally by email. Then one can review the access request and either approve or reject it.



Screenshot: Once approved, users of both tenants will see the ACL in the "Active" state, and soon Netris Agents will push the appropriate config throughout the switch fabric.



25.4 ACL Processing Order

1. User-defined Deny Rules
2. User-defined Permit Rules
3. Deny the rest

ROH (Routing on the Host)

To create more resilient and higher-performance data centers, some companies leverage the Linux ecosystem to run routing protocols directly on their servers. This is commonly known as ROH (Routing on the Host).

In ROH architectures, servers use a routing daemon to establish a BGP adjacency with the switch fabric on every physical link. ROH can run on bare metal servers, VMs, and even containers. The most commonly used routing daemon/suite is FRR.

Hosts connected to the network in ROH architecture don't have IP addresses on a shared Ethernet segment; instead an IP address is configured on the loopback interface and advertised over all BGP links towards switch fabric. This is a modern and optimal design, leveraging Layer-3 networking from the fabric to the servers.

By using only Layer-3 interfaces, Layer-2 protocols such as Spanning Tree (STP) can be minized and the reliability of the network increases.

The ROH architecture that is configured by Netris allows for leveraging ECMP load balancing capabilities of the switching hardware for the high-performance server load balancing (described in L3 Load Balancer section). For each instance of ROH, you'll need to create a ROH entry in Netris Controller.

26.1 Adding ROH Hosts

1. Navigate in the Netris UI to **Services→Instances (ROH)**
2. Click the **Add** button
3. Fill out the form based on the fields in the table below.
4. Click the **Add** button

Description of ROH instance fields:

- **Name** - Unique name for the ROH instance
- **Site** - Site where the current ROH instance belongs
- **Type** - Physical Server, for all servers forming a BGP adjacency directly with the switch fabric. Hypervisor, for using the hypervisor as an interim router. Proxmox is currently the only supported hypervisor.
- **ROH Routing Profile** - ROH Routing profile defines what set of routing prefixes to be advertised to ROH instances
    - **Default route only (most common design)** - Will advertise 0.0.0.0/0 + loopback address of the physically connected switch
    - **Default + Aggregate** - Will add prefixes of defined assignments + "Default" profile

    - **Full table** - Will advertise all prefixes available in the routing table of the connected switch
    - **Inherit** - Will inherit policy from site objects defined under Net→Sites
- **Legacy Mode** - Switch from default zero-config mode to using /30 IP addresses. Used for MSFT Windows Servers or other OS that doesn't support FRR.
- **+Port** - Physical Switch Ports anywhere on the network.
- **+IPv4** - IPv4 addresses for the loopback interface.
- **+Inbound Prefix List** - List of additional prefixes that the ROH server may advertise. Sometimes used to advertise container or VM networks.

Tip Many switches can't autodetect old 1Gbps ports. If attaching hosts with 1Gbps ports to 10Gpbs switch ports, you'll need to change the speed for a given Switch Port from Auto(default) to 1Gbps. You can edit a port in Net→Switch Ports individually or in bulk.

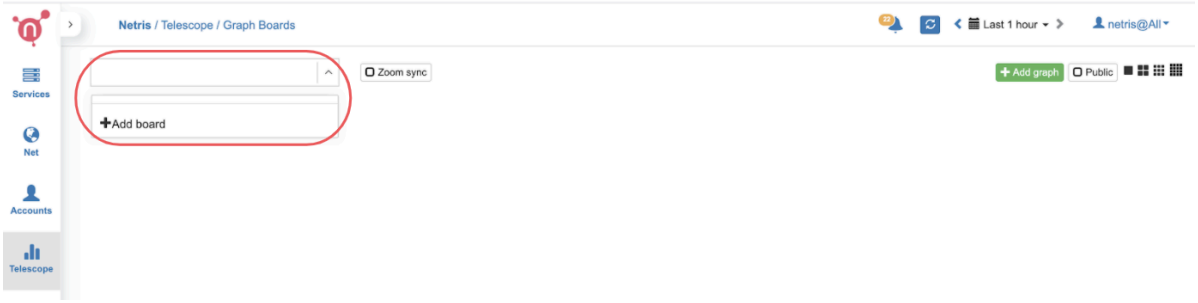
Visibility (Telescope)

27.1 Graph Boards

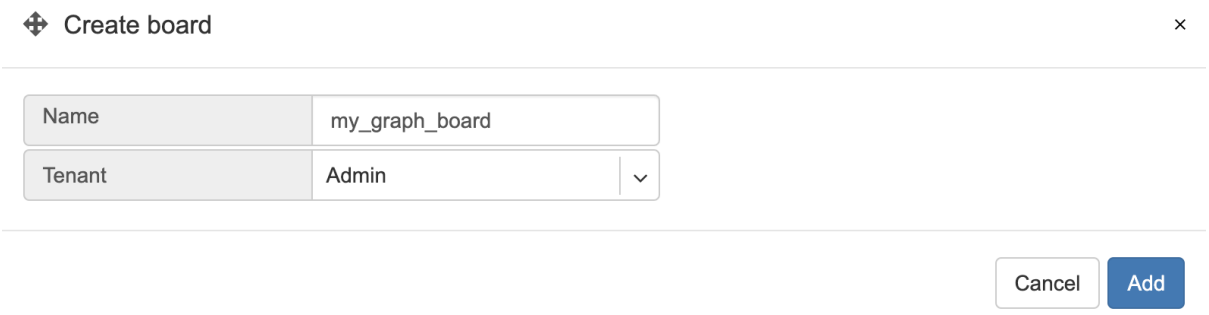
You can create custom graph boards with data sources available in different parts of the system. You can even sum multiple graphs and visualize them in a single view.

To start with Graph Boards, first, you need to add a new Graph Board.

1. Navigate to Telescope→Graph Boards, open the dropdown menu in the top left corner, then click +Add board.



2. Type a name and assign it to one of the tenants that you manage. Later on, you can optionally mark the Graph Board as public if you want the particular board to be visible to all users across multiple tenants.



Now you can add graphs by clicking +Add graph.

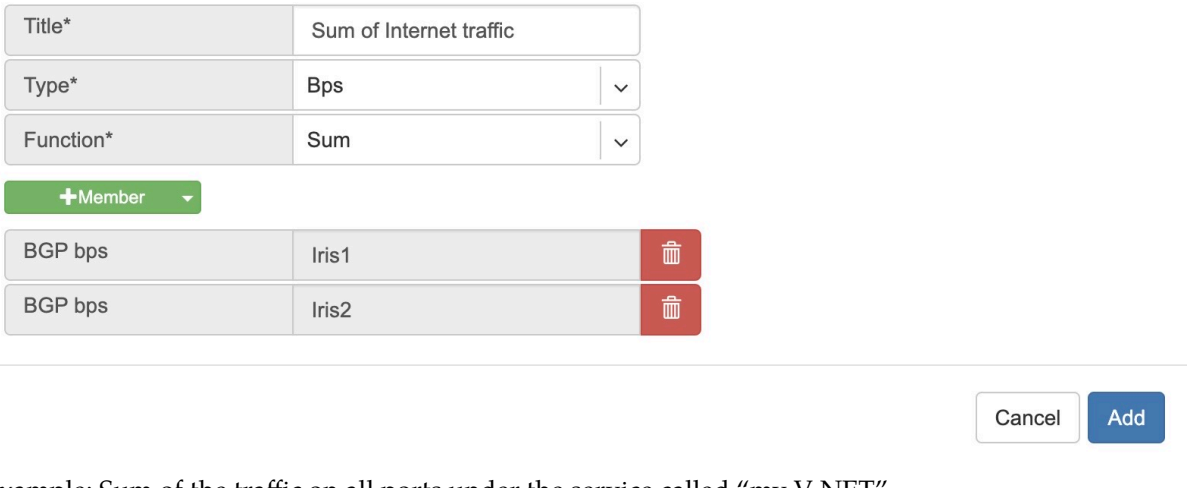
Description of +Add graph fields:

- **Title** - Title for the new graph.
- **Type** - Type of data source.

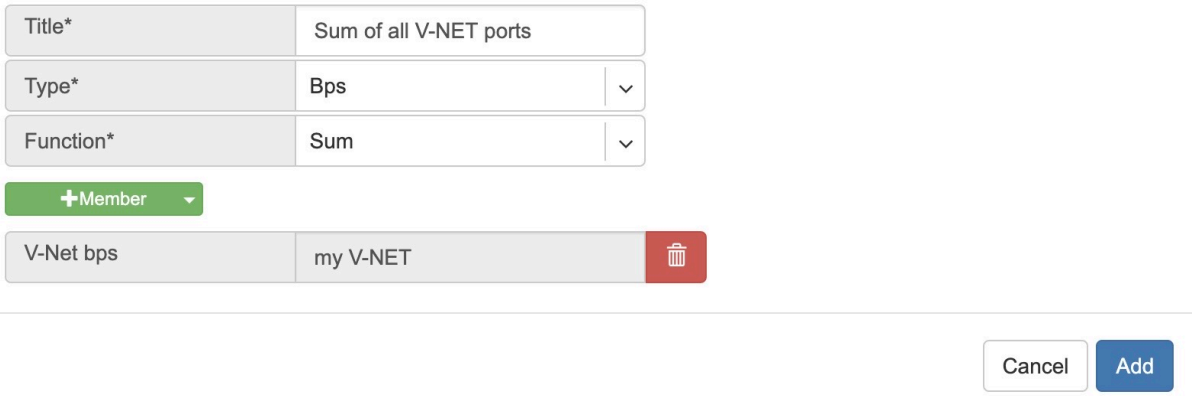
- Bps - Traffic bits per second.
- Pps - Traffic packets per second.
- Errors - Errors per second.
- Optical - Optical signal statistics/history.
- MAC Count - History of the number of MAC addresses on the port.

- **Function** - Currently, only summing is supported.
- **+Member** - Add data sources by service (E-BGP, V-NET, etc..) or by Switch Port.

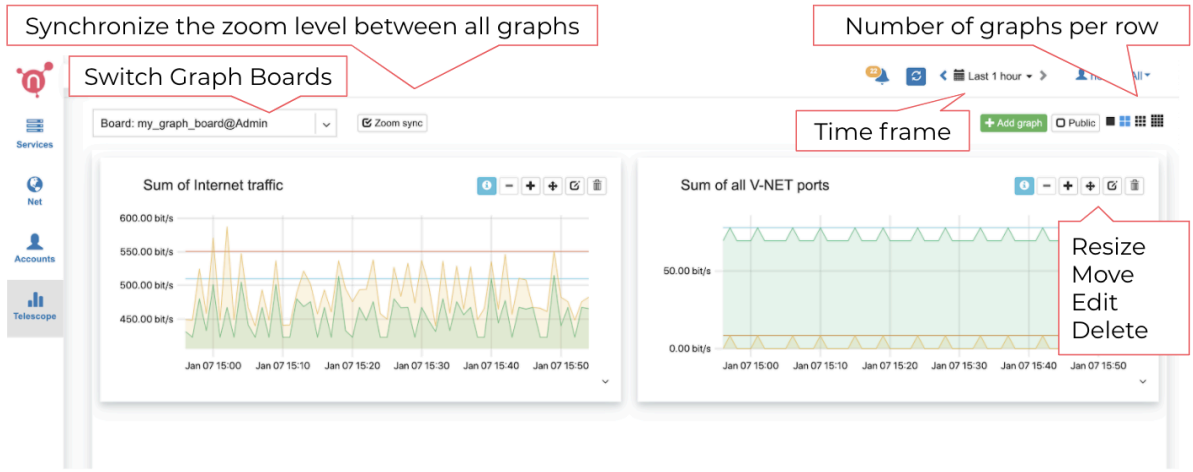
Example: Sum of traffic on two ISP(Iris1 + Iris2) links.



Example: Sum of the traffic on all ports under the service called “my V-NET”



Screenshot: Listing of a Graph Board with the explanation of the controls.



27.2 API Logs

Comprehensive logging of all API calls sent to Netris Controller with the ability to search by various attributes, sort by each column, and filter by method type.

27.3 Dashboard

Netris, besides automatic configuration, also provides automatic monitoring of the entire network without the need for configuration of the monitoring systems.

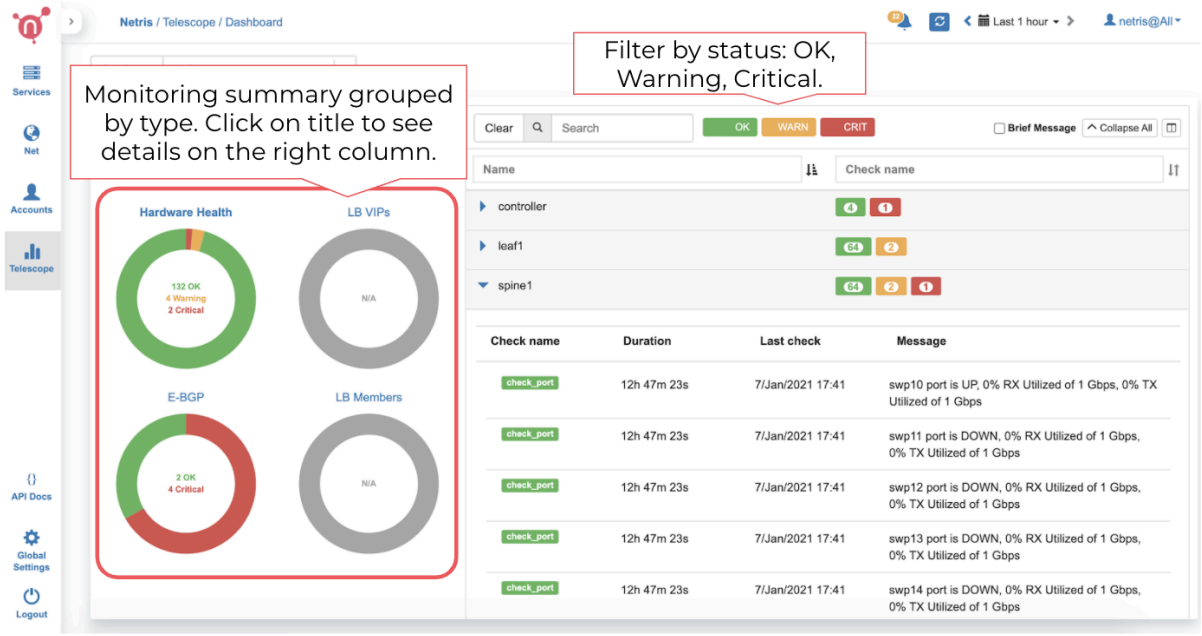
Telescope→Dashboard summarizes Network Health, which can also be accessed by clicking on the Netris icon in the top left corner.

Description of the pie charts.

- **Hardware Health** - summary of CPU, RAM, disk utilization. Statuses of power supplies, fans, temperature sensors, critical system services, and time synchronization. Statuses of switch port link, utilization, optical signal levels, and BGP sessions.
- **E-BGP** - Statuses of external BGP sessions.
- **LB VIP** - Statuses of Load Balancer frontend / VIP availability.
- **LB Members** - Statuses of Load Balancer backend members.

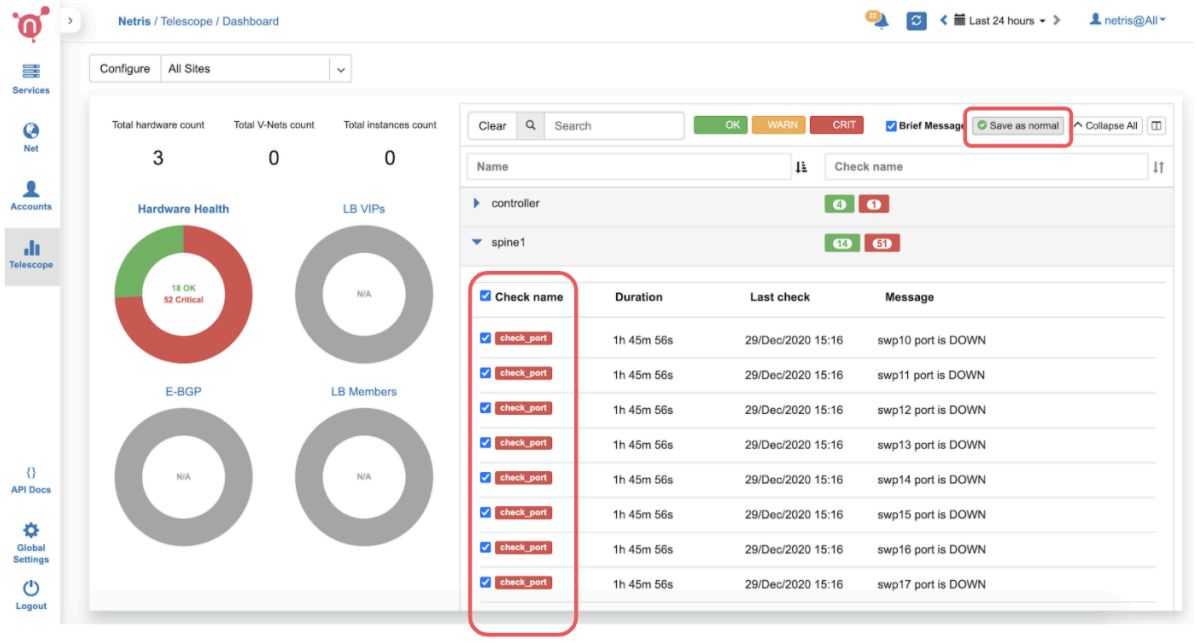
By clicking on each title you can see the details of the checks on the right side.

Screenshot: Dashboard showing details of “Hardware Health.”



Port up/down state can be set to "Save as normal." So the system will alarm only if the actual state is different from the saved as the normal state.

Screenshot: "Save as normal" on selected ports.



Accounts

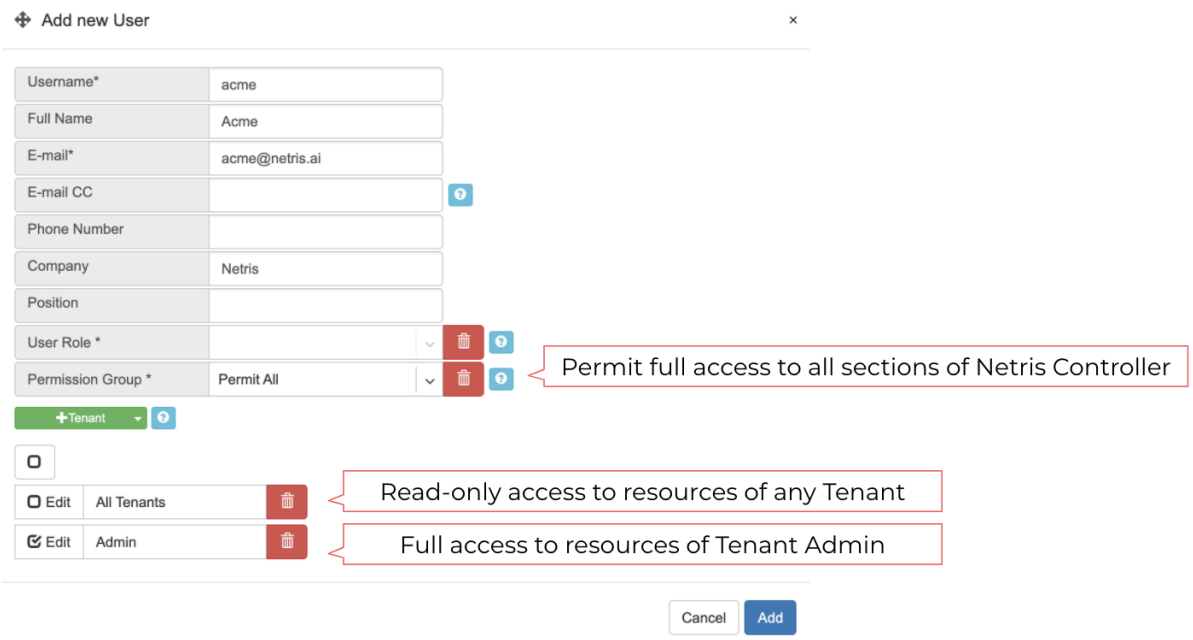
The accounts section is for the management of user accounts, access permissions, and tenants.

28.1 Users

Description of User account fields:

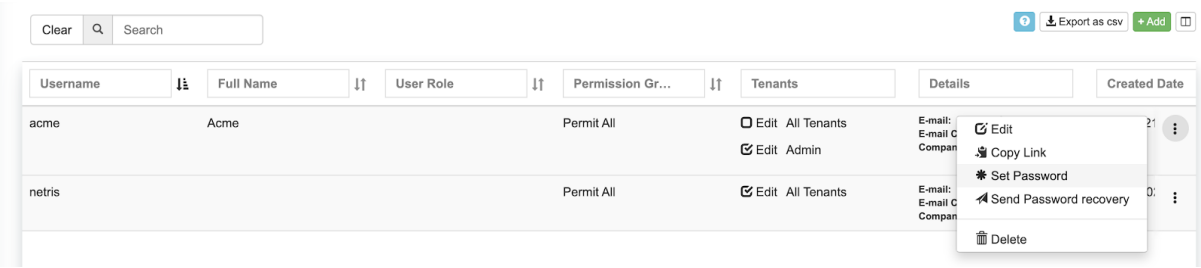
- **Username** - Unique username.
- **Full Name** - Full Name of the user.
- **E-mail** - The email address of the user. Also used for system notifications and for password retrieval.
- **E-mail CC** - Send copies of email notifications to this address.
- **Phone Number** - User's phone number.
- **Company** - Company the user works for. Usually useful for multi-tenant systems where the company provides Netris Controller access to customers.
- **Position** - Position within the company.
- **User Role** - When using a User Role object to define RBAC (role-based access control), Permissions Group and Tenant fields will deactivate.
- **Permission Group** - User permissions for viewing and editing parts of the Netris Controller. (if User Role is not used)
- **+Tenant** - User permissions for viewing and editing services using Switch Port and IP resources assigned to various Tenants. (if User Role is not used)

Example: Creating a user with full access to all sections of Netris Controller, read-only access to resources managed by any Tenant, and full access to resources assigned to the Tenant Admin.



Password: To set a password or email the user for a password form, go to the listing of usernames and click the menu on the right side.

Example: Listing of user accounts.

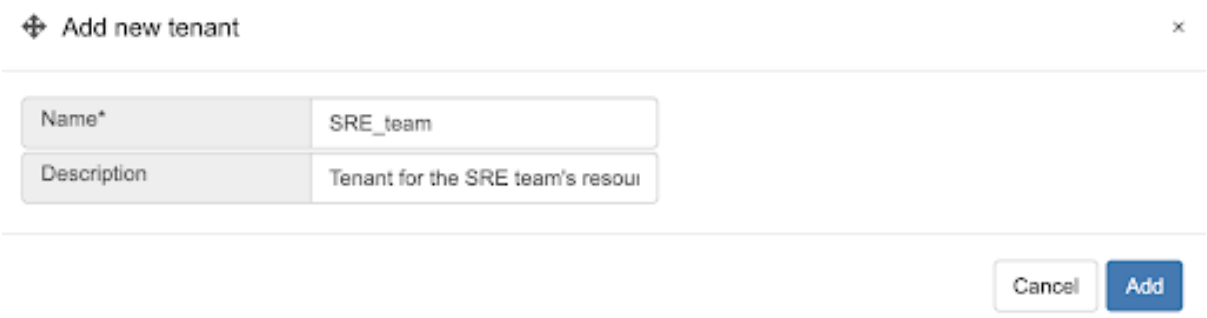


28.2 Tenants

IP addresses and Switch Ports are network resources that can be assigned to different Tenants to have under their management. Admin is the default tenant, and by default, it owns all the resources. The concept of Tenants can be used for sharing and delegation of control over the network resources, typically used by network teams to grant access to other teams for requesting & managing network services using the Netris Controller as a self service portal or programmatically (with Kubernetes CRDs) as part of DevOps/NetOps pipeline.

A Tenant has just two fields, the unique name and custom description.

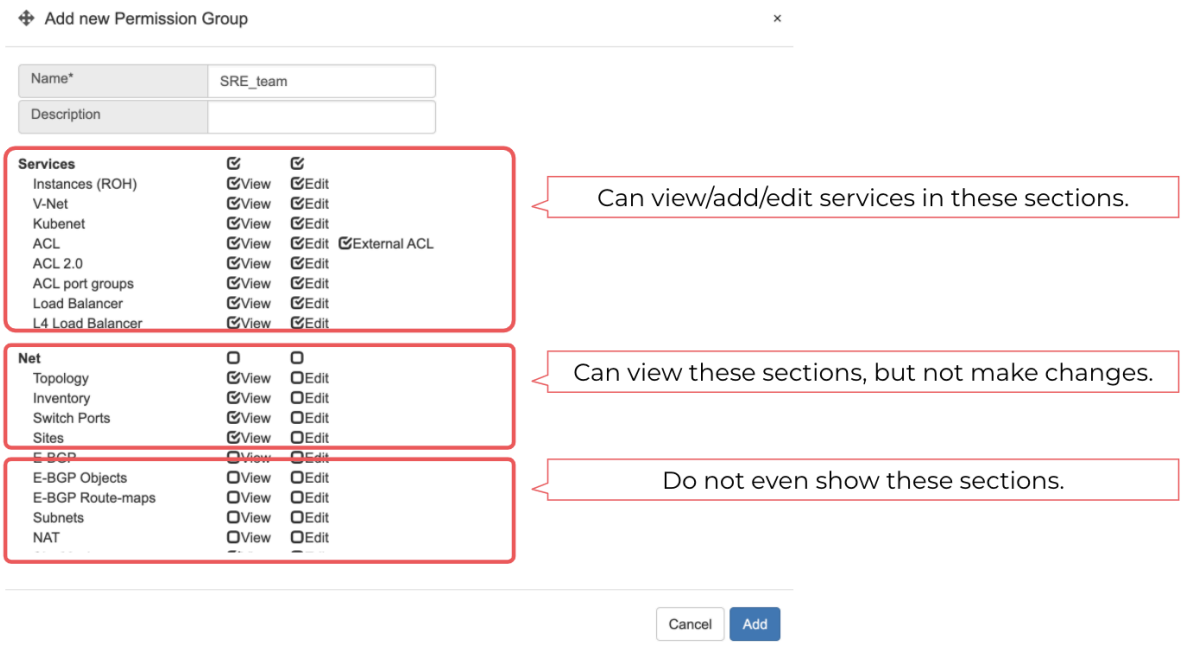
Example: Adding a tenant.



28.3 Permission Groups

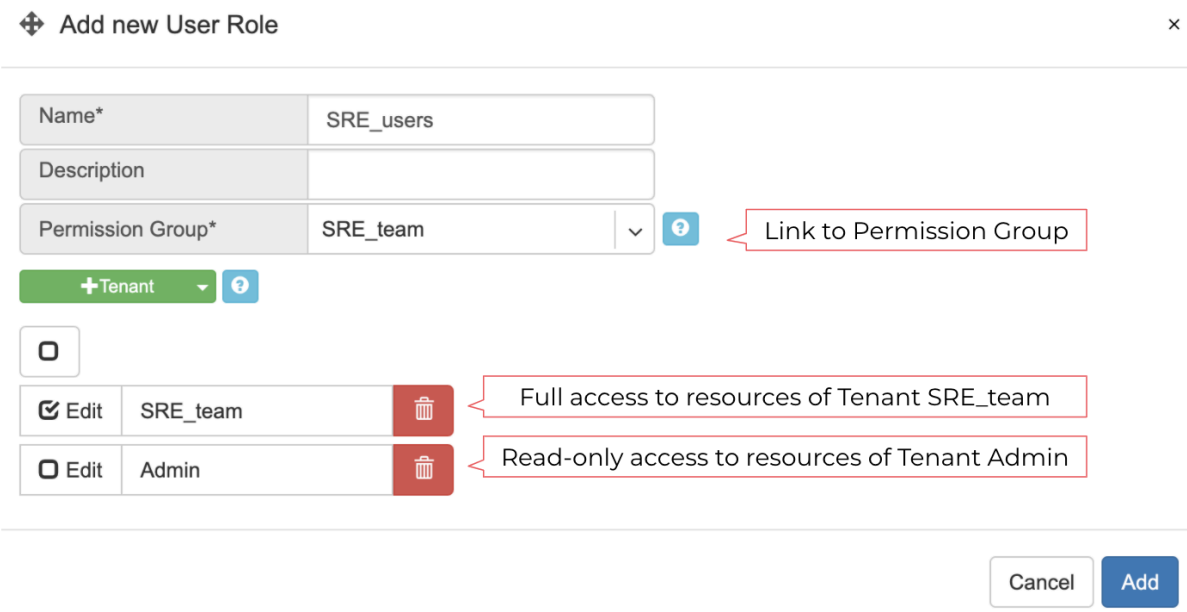
Permission Groups are a list of permissions on a per section basis that can be attached individually to a User or a User Role. Every section has a View and Edit attribute. The view defines if users with this Permission Group can see the particular section at all. Edit defines if users with this Permission Group can edit services and policies in specific sections.

Example: Permission Group.



28.4 User Roles

Permission Groups and Tenants can be either linked directly to an individual username or can be linked to a User Role object which then can be linked to an individual username.


Add new User Role
Name*
SRE_users
Description
Permission Group*
SRE_team
Link to Permission Group
+Tenant
Edit
SRE_team
Full access to resources of Tenant SRE_team
Edit
Admin
Read-only access to resources of Tenant Admin
Cancel
Add

Release notes

- DPDK data plane support for SoftGate nodes. - Provides higher SoftGate performance. Up to 27Mpps, 100Gbps for L3 routing, 12Mpps with NAT rules on.
- L4 Load Balancer. - In addition to switch-based Anycast Load Balancer, we now support a SoftGate/DPDK-based L4 Load Balancer. L4LB integrates with Kubernetes providing cloud-like load balancer service (type: load-balancer).
- Kubenet - a network service purpose-built for Kubernetes cluster nodes. Kubenet integrates with Kube API to provide an on-demand load balancer andother Kubernetes specific networking features. Netris Kubenet is designed to complement Kubernetes CNI networking with modern physical networking.
- API logs - Comprehensive logging of all API calls sent to Netris Controller with the ability to search by various attributes, sort by each column, and filter by method type.
- Site Mesh - a Netris service for automatically configuring site-to-site interconnect over the public Internet. Site Mesh supports configuration for WireGuard to create encrypted tunnels between participating sites andautomatically generates configuration for FRR to run dynamic routing. In a few clicks, services in one site get connectivity to services in other sites over a mesh of WireGuard tunnels.
- Ubuntu/SwitchDev updates - Removed the requirement for a hairpin loop cable. Removed the need for IP address reservation for V-NET, switching entirely to the anycast default gateway.
- Controller distributions - Netris controller, is now available in three deployment forms. 1) On-prem KVM virtual machine. 2) Kubernetes application. 3) Managed/Hosted in the cloud.
- Inventory Profiles - A construct for defining access security, timezone, DNS,NTP settings profiles for network switches and SoftGate nodes.
- Switch/SoftGate agents - New installer with easy initial config tool. Support for IP and FQDN as a controller address. Authentication key.
- GUI - Improved Net→Topology section, becoming the main and required place for defining the network topology. All sections got a column organizer, so every user can order and hide/show columns to their comfort.

SoftGate Performance

The following tested results are offered to help properly size the hardware needed for a SoftGate with various types of services:

Table 30.1. SoftGate Performance

	Routing(mpps)	**Routing with unmatched DNAT rules (mpps)**	**Routing with unmatched DNAT and SNAT rules (mpps)**	**DNAT**	**SNAT**	DNAT with unmatched SNAT rules
Xeon E5-2660 CPU cores: (6 core for fw) Scheduler 0 Forward 1-7 KNI 8-10 OS 11-15	~27mpps	~17mpps	7mpps	9mpps	~2mpps	
Xeon Gold 6130 CPU cores: (6 core for fw) Scheduler 0 Forward 3-13(odd numbers) KNI 15 17 19	~31mpps	~22mpps	~15mpps	~11mpps	~3mpps(3M conntrack count) ~7mpps(20k conntrack count)	~11mpps
Xeon Gold 6130 CPU cores: (8 core for fw) Scheduler 0 Forward 3-17(odd numbers) KNI 19 21 23	~38mpps	~28mpps	19mpps	12mpps		~12mpps