Nutanix Virtual Sensor (vSensor) Deployment Guide

Version: Nov 6, 2025

Table of Contents

© 2025 Vectra AI, Inc.

Introduction

This guide is intended to help customers or partners deploy virtual Sensors (vSensors) in Nutanix environments and pair them to your Vectra Brain. It will cover basic background information, connectivity requirements (firewall rules that may be needed in your environment), deployment of the vSensor, and pairing.

vSensors behave much in the same way that physical Sensors do. One advantage is that there is no cost to deploy a vSensor other than your own costs to provide and maintain the infrastructure they run in. vSensors also allow you to capture and analyze traffic that only exists in the virtual environment.

Please note the following details:

- Please see Nutanix Traffic Capture Options for additional guidance on choosing between Service Chaining 1.0 and Traffic mirroring to steer traffic to your Vectra vSensor.
    - The steps to deploy the Vectra vSensor will be different depending on your traffic steering method.
- For Service Chaining 1.0
    - Capture of virtual traffic is done by a network function NIC that must be added at the ACLI and uses Prism Central REST API calls to create and direct subnets to the NIC via a network function chain.
    - Only VLAN Basic subnets are supported. NCVLANs (Network Controller VLANs or VLAN subnets) are not supported with service chaining 1.0.
    - Nutanix has a public KB that describes service chain integration at a high level.
        - Vectra does not automate network function chain integration on behalf of the customer.
    - Only TAP mode is supported for the network function chain.
        - Inline mode (unsupported) is typically used for firewall type devices that are inline.
    - This guide will provide steps that can be used to create the service chain for vSensor deployment in TAP mode and direct an entire AHV network (including VM to VM traffic) to the vSensor.
        - Other traffic direction options such as using flow security policies or directing a single VM NIC to the vSensor are not covered in this guide but are also possible.
    - **It is strongly recommended, by both Nutanix and Vectra, that if you are not already familiar with network function chain implementation using Prism Central REST API calls, you should engage the Nutanix account team to evaluate the exact requirements and work with Nutanix Services to assist with manual or scripted network function chain implementation.**
- For Traffic Mirroring
    - Both VLAN Basic subnets and NCVLANs are supported.
    - Physical network traffic capture is possible by mirroring a host port or bonded host port on a node.
    - Traffic mirroring requires a Mirror Destination NIC to be added to the vSensor for traffic capture (not a network function NIC). A traffic mirroring session would use this capture NIC as the destination.

Nutanix vSensors can be used in both Respond UX and Quadrant UX deployments. For more detail on Respond UX vs Quadrant UX please see Vectra Analyst User Experiences (Respond vs Quadrant). One of the below guides should be the starting point for your overall Vectra deployment:

- Vectra Respond UX Deployment Guide
- Vectra Quadrant UX Deployment Guide

Either of the above guides cover basic firewall rules needed for the overall deployment and initial platform settings. Virtual Sensor (VMware, Hyper-V, KVM, AWS, Amazon, and GCP) configuration and pairing and covered in their respective guides. Physical appliance pairing is covered in the Vectra Physical Appliance Pairing Guide. Please see the Vectra Product Documentation Index on the Vectra support site for additional documentation including deployment guides for CDR for M365 / IDR for Azure AD and CDR for AWS.

For customers who run VMware vSphere ESXi hypervisors in Nutanix AOS environments, this is a supported configuration. Please see Support for Nutanix Management of VMware vSphere (ESXi) for details.

© 2025 Vectra AI, Inc.

Nutanix Traffic Capture Options

Nutanix Links, Terminology, and Facts

- Flow Virtual Networking Guide (pc.2024.2)
- Flow Network Security Next-Gen Version 5.2.x Guide
- Nutanix Bible - Network Controller

Flow Virtual Networking, powered by Network Controller, is a software-defined networking solution that provides multi-tenant isolation, self-service provisioning, and IP address preservation using VPCs, subnets, and other virtual components that are separate from the physical network, for the AHV clusters.

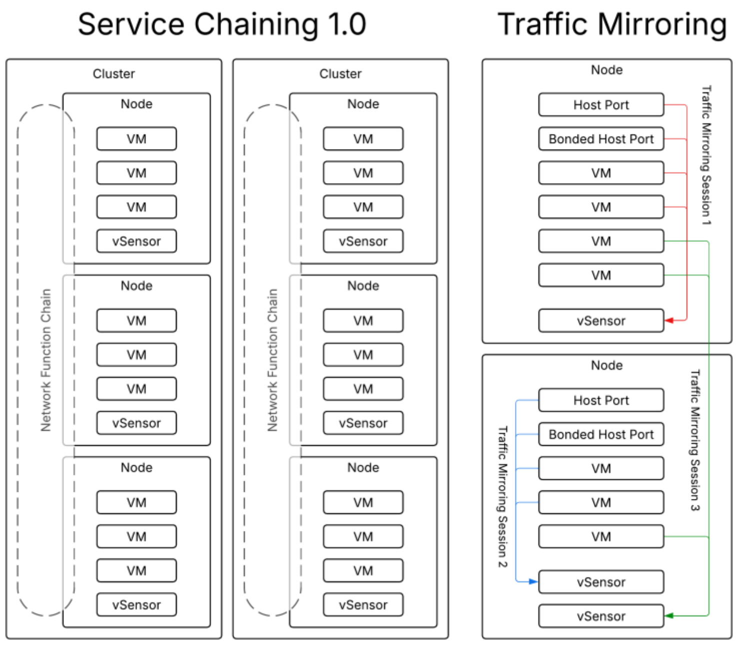
The Network Controller is the networking component of Prism Central that manages and controls configuration, monitoring and optimization of network resources for Flow Virtual Networking VPCs and VLAN subnets. It provides programmability, automation, and centralized control for configuring and managing network flows.

Network Controller is necessary to use centralized VLAN management, Flow Virtual Networking and Flow Network Security Next Generation.

Subnet types and “Single Stack” and “Dual Stack” Nutanix networking categories:

- VLAN Subnets
  - Network Controller managed VLANs (NCVLANs). These can only be managed in Prism Central.
- VLAN Basic Subnets
  - Managed by the Acropolis leader of their Prism Element cluster. These are traditional AHV VLANs
- Single Stack – uses only NCVLAN subnets.
- Dual Stack – allows both NCVLAN subnets and VLAN Basic subnets.
  - Flow Network Security 4.2.0 and 4.2.1 are the last versions that support Dual Stack networking.
    - If your version is newer, Service Chaining 1.0 cannot be used to steer traffic.
  - To see your version navigate in Prism Central to *Admin Center* > *LCM* > *Prism Central Cluster.*

Nutanix Traffic Steering Options



© 2025 Vectra AI, Inc.

Nutanix has two primary steering options that can be used to direct traffic to the capture port of a vSensor:

- **Service Chaining 1.0**
    - Also known as “Service Chain Insertion”.
    - Uses a “Network Function Chain” to TAP (copy out of band) subnet traffic seen on a cluster to Vectra vSensors deployed on each node in the cluster.
    - Advantages
        - Captures VM to VM traffic on the same node and any traffic that is seen on the cluster for the subnets that are mapped to it. In other words, you get E/W between VMs on the node and N/S in and out of the node for any mapped subnet.
    - Disadvantages
        - Requires “Dual Stack” Nutanix networking.
        - Does NOT support NCVLAN subnets and only supports VLAN Basic subnets.
        - Configuration is more complicated.
    - **Please Note!**
        - A single network function chain (NFC) is required per Nutanix cluster in your environment.
        - A Vectra vSensor should be deployed on each node of the cluster.
        - Subnets will be mapped to each network function chain.
            - Any subnet that will be seen on a node in a cluster should be mapped to the NFC.
            - This means that a subnet should be mapped to multiple NFCs when you have more than one cluster in your environment that will see traffic from the same subnet.
        - Traffic seen on any node that is in a mapped subnet will then be copied via the NFC to the Vectra vSensor capture port that was setup as a network function NIC.
- **Traffic Mirroring**
    - Uses “Traffic Mirroring Sessions” to map host (node) ports, bonded host ports, or VM interfaces to “Mirror Destination NICs” (Vectra vSensor capture ports). You get E/W between any VMs in the mirroring session, and N/S in and out of the node if you mirror a host port or bonded host port.
    - Advantages
        - Can capture physical network traffic that is seen on a host port or bonded host port.
        - Works in both “Dual Stack” and “Single Stack” Nutanix networking environments.
        - Supports both NCVLAN subnets and VLAN Basic subnets.
        - Configuration is simpler and fully done in the Prism Central UI.
    - Disadvantages
        - By default, only supports 4 sources in a traffic mirroring session.
            - Nutanix support or professional services (PS) can raise the 4-source limit to as high as 50 if there is sufficient headroom in the environment. **Please engage Nutanix support or PS to request increases to the 4-source limit.**
        - Managing traffic mirroring sessions can become burdensome.
            - You must specify every source specifically and cannot just map a subnet to a vSensor capture port. For large numbers of VMs it can become complicated to properly map all VMs.
    - **Please Note!**
        - A traffic mirror destination NIC can also only be used as a destination in one traffic mirroring session.
        - A host port or bonded host port does not support mirroring to a remote host. This means that the vSensor must be on the same node (AHV host) as the source host/bonded port.
            - See Traffic Mirroring Sessions 1 and 2 in the diagram above.
        - VM interfaces can be mirrored to a remote host. This means that the vSensor can be on a different node (AHV host) than the source VM.
            - See Traffic Mirroring Session 3 in the diagram above.
        - For more details on Traffic Mirroring, please see the link.

© 2025 Vectra AI, Inc.

About Nutanix vSensor Images

The Brain makes an image available in QCOW2 format for download and subsequent use while deploying Nutanix vSensors. Vectra appliances typically operate with updates enabled. Regular updates ensure that the appliances are running the very latest version. Deployed Sensors and vSensors also update regularly from the Brain. Once a vSensor has been deployed and paired to the Brain, it will be updated from and stay current with its paired Brain.

- Please note that as your Vectra Brain is updated, the image for a Nutanix vSensor is also updated.
    - If you deploy additional Nutanix vSensors in the future, always download a fresh copy of the image from an up-to-date Brain to ensure you are working with the latest code.

Nutanix vSensor Requirements and Throughput

Vectra supports Nutanix for customers choosing to deploy virtual Sensors (vSensors) to capture virtual traffic. As of v9.6 Vectra also supports deployment of a Nutanix Brain.

Nutanix vSensor Configurations

Nutanix Version Requirements	Capture Ports	Cores	Memory	Storage	Performance
All configurations need at least: AOS: 5.20.3.5 / AHV: 20201105.2267 **(Note AOS 6.8 is incompatible, while 6.8.1 and higher of 6.8 branch is ok)**	1	2	8 GB	100 GB	500 Mbps
	1	4	8 GB	150 GB	1 Gbps
	1	8	16 GB	150 GB	2 Gbps
	1	16	64 GB	500 GB	5 Gbps

- Nutanix vSensors also require a management port that is separate from the capture port.
- Vectra recommends that Sensors are configured to use storage local to the hypervisor and are not stored on a SAN. Vectra vSensors require extremely high throughput from their disk storage and this throughput cannot normally be sustained by SAN systems without impact to other SAN users.

Connectivity Requirements

The Vectra Respond UX Deployment Guide or Vectra Quadrant UX Deployment Guide detail basic connectivity requirements for initial platform deployment. It also gives guidance on firewall/proxy SSL inspection, Internet access to and from the Vectra Brain, and guidance for air-gapped environments. For full detail on all possible firewall rules that might be required in your environment, please see the Firewall Requirements for Vectra Appliances KB.

Connectivity Requirements for Nutanix vSensors

Source	Destination	Protocol/Port	Description
Admin Hosts	vSensors	TCP/22 (SSH)	CLI access to vSensor
Brain	vSensors	TCP/22 (SSH)	Remote management and troubleshooting
vSensors	Brain	TCP/22 (SSH) TCP/443 (HTTPS)	Pairing, metadata transfer, and ongoing communication

© 2025 Vectra AI, Inc.

Please note:

- vSensors do not communicate with the Vectra Cloud.
- All communication sessions with vSensors are initiated from the vSensor to the Brain.
- Updates for vSensors are downloaded to the Vectra Brain and the vSensor retrieves them from the Brain.
- Command line access to the vSensor can also be obtained via the console in your hypervisor.

Preparing to Deploy Nutanix vSensors

Some basic information will need to be gathered or known prior to beginning deployment of a Nutanix vSensor.

- IP address and subnet mask for the Management interface of the vSensor. DHCP is also supported.
- IP address or hostname of the Vectra Brain that you will be pairing with.
  - This will be baked into the vSensor image downloaded from your Vectra UI and will point to the Brain associated with your Vectra UI.
  - Whether this points to the IP of the Brain or the hostname depends on what you have set in your Brain under *Data Sources > Network > Sensors > Sensor Configuration > Sensor Registration and Pairing*". If a DNS Name (host name) is configured in *Data Sources > Network > Brain Setup > Brain*, you will see a radio button to choose either "**DNS Name"** or **"Management IP Address"**.
- DNS server addresses.
- To configure your vSensor, you will need access to the vSensor Command Line Interface (CLI) either via the console in your hypervisor or via SSH.
  - DHCP is enabled by default upon vSensor initial boot.
  - You must know the IP that was assigned via DHCP to SSH to the CLI, otherwise you will need to use the hypervisor console to access the CLI and perform initial configuration.
- For production monitoring, ensure that the vSensor VM is kept running 24x7, and ensure that the hypervisor does not overcommit resources or otherwise misrepresent the resources it is providing to the vSensor.

On the Nutanix side of the configuration, please keep in mind the following.

- You must have access to both Prism Central and Prism Element for Service Chaining 1.0
  - You must have permissions to make Prism Central REST API calls for Service Chaining 1.0.
- Traffic Mirroring only requires the Prism Central UI for deployment.
  - Setting the vSensor as an Agent VM will require Prism Element or ACLI commands.
- You must have access to a Controller VM (CVM) ACLI command line to perform ACLI commands.

Choosing a Deployment Method

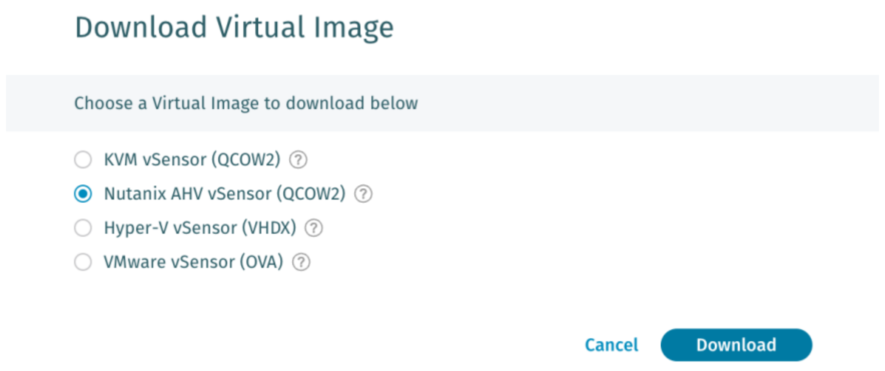
- Both deployment methods will share the Pre-Deployment Steps for the Nutanix vSensor.
- You can use both deployment methods if you desire when deploying multiple Nutanix vSensors.
  - For example, you might use one vSensor deployed using Traffic Mirroring to capture physical network traffic that arrives on a host port or host bonded port on a particular node and another vSensor using Service Chaining 1.0 to capture VM to VM traffic and all other subnet traffic.
  - More than one vSensor can be deployed on a single node in your Nutanix cluster if required.
- Nutanix vSensor Deployment for Service Chaining 1.0
- Nutanix vSensor Deployment for Traffic Mirroring
- For either deployment method, ensure you follow the Shared Steps After vSensor Deployment.

© 2025 Vectra AI, Inc.

Pre-Deployment Steps for the Nutanix vSensor

1. Downloading the Nutanix vSensor Image

The Nutanix vSensor image is available under *Data Sources* > *Network* > *Sensors* in your Vectra UI. Navigate to this area and then click “Download Virtual Image” at the top right and select the Nutanix vSensor (QCOW2) option.



Download Virtual Image

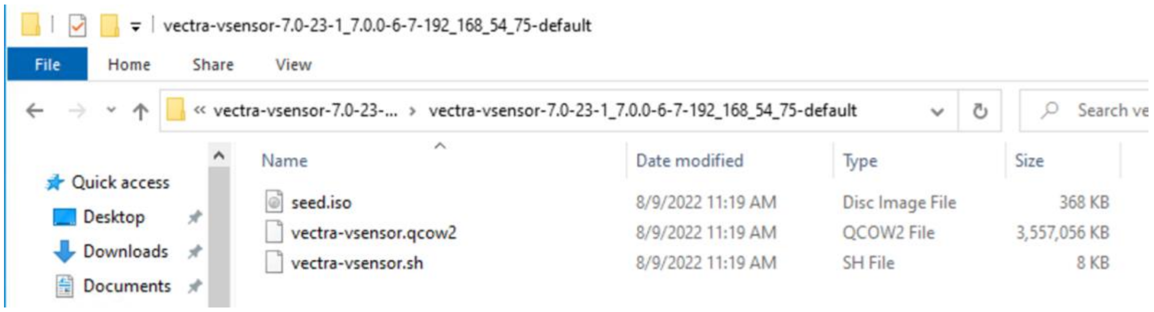
Choose a Virtual Image to download below

- KVM vSensor (QCOW2)
- [✓ Nutanix AHV vSensor (QCOW2)]
- Hyper-V vSensor (VHDX)
- VMware vSensor (OVA)

2. Extract vSensor Image Files

The downloaded vSensor image will have a filename similar to this (updates to the image after this documentation was written will change this): “vectra-vsensor-117678-gee9a02c9f7-1_9.0.0-18-57-192_168_52_73-default.tar.gz”

Using the unarchiving tool of your choice, decompress the .gz file. Then untar the file to a directory of your choice. That directory will contain 3 files such as:



vectra-vsensor-7.0-23-1_7.0.0-6-7-192_168_54_75-default

File
Home
Share
View
« vectra-vsensor-7.0-23-... › vectra-vsensor-7.0-23-1_7.0.0-6-7-192_168_54_75-default
Search ve
Quick access
Desktop
Downloads
Documents

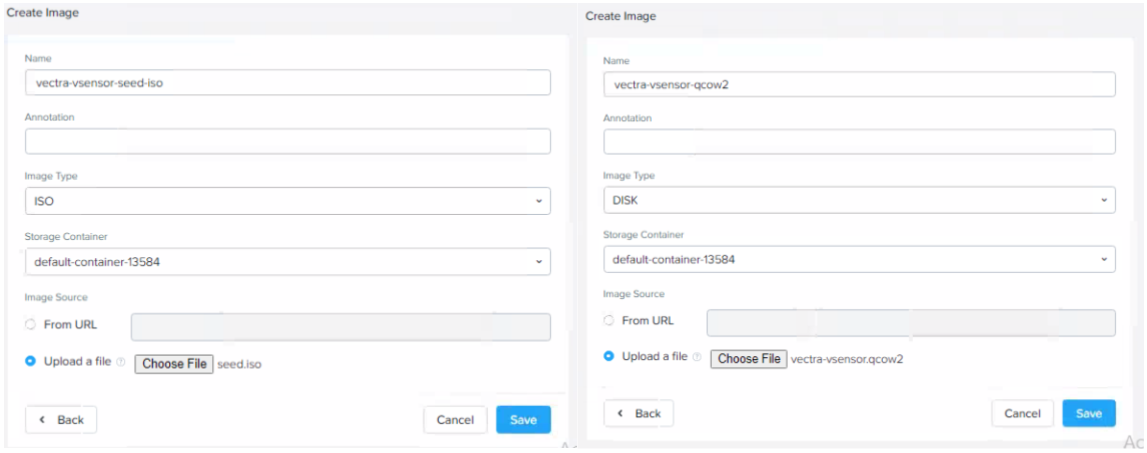
Name	Date modified	Type	Size
seed.iso	8/9/2022 11:19 AM	Disc Image File	368 KB
vectra-vsensor.qcow2	8/9/2022 11:19 AM	QCOW2 File	3,557,056 KB
vectra-vsensor.sh	8/9/2022 11:19 AM	SH File	8 KB

The “vectra-vsensor.sh” file can be ignored for Nutanix vSensor deployment. The base image is shared for both Nutanix and KVM deployment. KVM deployment uses that file, but Nutanix only needs the .iso and qcow2 files.

3. Upload vSensor Image Files to Nutanix

If using Prism Element for the initial VM deployment (Service Chaining 1.0 Only):

Using the image service (available in Prism Element at *Settings* > *Image Configuration* > *"+ Upload Image"*), upload the "seed.iso" and "vectra-vsensor.qcow2" images. You can name these how you prefer in Nutanix.



Create Image

Name: vectra-vsensor-seed-iso
Annotation:
Image Type: ISO
Storage Container: default-container-13584
Image Source: ○ From URL [✓ Upload a file] Choose File seed.iso

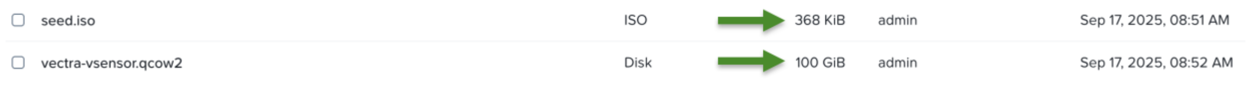
Create Image

Name: vectra-vsensor-qcow2
Annotation:
Image Type: DISK
Storage Container: default-container-13584
Image Source: ○ From URL [✓ Upload a file] Choose File vectra-vsensor.qcow2

- Please note that the seed.iso should be set to an "Image Type" of ISO and the .qcow2 should be Disk.
- Once the images have become "Active", you can proceed to create the VM. Please note that the .qcow2 image will take considerably longer to become active than the seed.iso.

If using Prism Central for the initial VM deployment (for Service Chaining 1.0 and Traffic Mirroring):

Navigate to *Infrastructure* > *Compute* > *Images* > *"Add Image"*, and upload the "seed.iso" and "vectra-vsensor.qcow2" images. The "Image Source" should always be "Image File" and when you select the seed.iso or qcow disk file they will automatically be recognized as ISO or DISK Image types. Unlike when using PE to upload the images, you do **NOT** need to wait for the images to become "Active", but you do need to wait for them to finish uploading. When you see the file sizes appear in the image library, you know they are ready to deploy:



seed.iso	ISO	368 KiB	admin	Sep 17, 2025, 08:51 AM
vectra-vsensor.qcow2	Disk	100 GiB	admin	Sep 17, 2025, 08:52 AM

Examples of the upload dialogs for Prism Central:

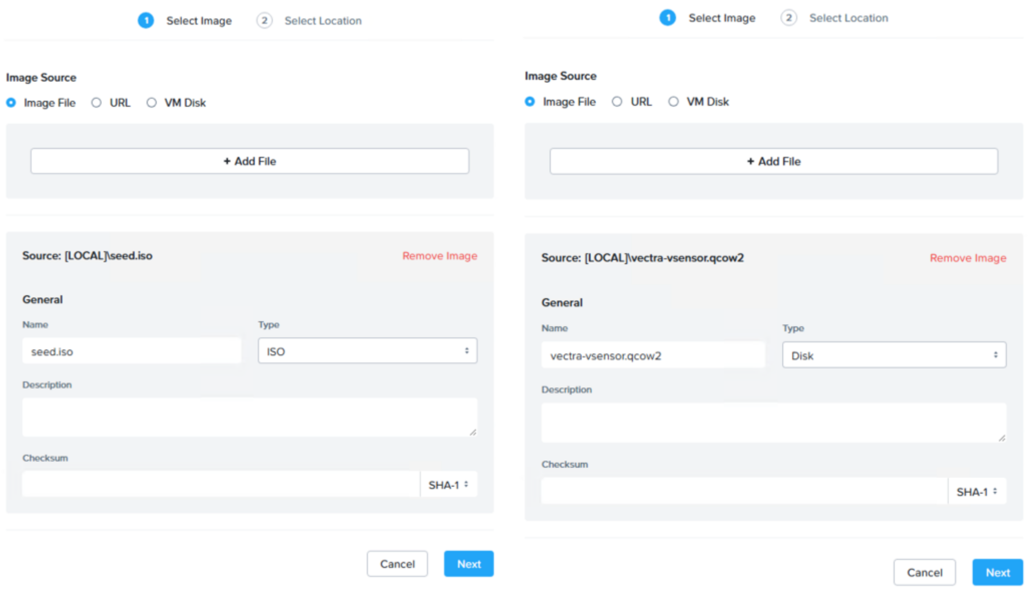


Image Source: [✓ Image File] ○ URL ○ VM Disk

Source: [LOCAL]\seed.iso

General
Name: seed.iso
Type: ISO
Description:
Checksum: SHA-1

Image Source: [✓ Image File] ○ URL ○ VM Disk

Source: [LOCAL]\vectra-vsensor.qcow2

General
Name: vectra-vsensor.qcow2
Type: Disk
Description:
Checksum: SHA-1

Nutanix vSensor Deployment for Service Chaining 1.0

Guidance for Using the Nutanix Prism Central REST API

In many of the following deployment steps we will be showing example Prism Central REST API v3 calls. Additional guidance for using the API can be found at the following links:

- GET /api/nutanix/understanding-rest-api
- Nutanix API versions – What are they and what does each one do?
- Accessing the REST API Explorer

Most customers find it easier to use dedicated API tools such as Postman or Insomnia to make API calls. API calls can also be made using curl or built into code for programmatic execution. Vectra does not endorse any specific tool for making the calls. Vectra will provide the type of REST call (GET, POST, DELETE, etc), endpoint, and instructions or body examples in this guide for execution using the tool of your choice.

Vectra provides a .json Postman collection that can be imported as an attachment to the Nutanix vSensor Deployment Guide KB. This will need to be modified to add your Basic Auth credentials. Also the Prism Central IP or Hostname {{pc_hostname_or_ip}} variable will need to be configured with the IP or hostname of your Prism Central deployment. We will highlight important pieces of data that may also need to be used in subsequent calls and highlight where we used them.

Some of the calls required to complete the configuration will return a “status” section with a “state” of “PENDING” and a “task_uuid”. To check the status of these tasks a GET operation can be executed against the /tasks endpoint. Below is an example response showing this along with a task check and response.

Abbreviated response example:

```
{
    "status": {
        "state": "PENDING",
        "execution_context": {
            "task_uuid": "3acead7a-bfec-4adb-b75c-733f191631ef"
        }
    }
.
.
.
```

Task status check request:

```
GET - https://192.168.50.9:9440/api/nutanix/v3/tasks/3acead7a-bfec-4adb-b75c-733f191631ef
```

Response example:

```
{
    "status": "SUCCEEDED",
    "last_update_time": "2022-08-22T19:05:36Z",
    "logical_timestamp": 2,
    "entity_reference_list": [
        {
            "kind": "network_function_chain",
            "uuid": "24936fa3-54e5-4ae2-a80c-411fae635236"
        }
    ],
.
.
.
```

Service Chaining 1.0 Deployment Overview

Before proceeding with the deployment, it is important to understand that certain parts of the deployment MUST happen in order. Some steps will have multiple options to complete the steps and alternative methods may be mentioned but not detailed. For example, images can be uploaded to your Nutanix environment using both Prism Element (PE) or Prism Central (PC), but PE and PC maintain separate image libraries. If the image was uploaded via PE, you must deploy the vSensor VM via PE. Significant differences for alternate methods will be called out.

Below is an overview of the major steps:

1. Create vSensor VM
    a. Add only the standard NIC that will be used for management (MGT1) at this time, the network function NIC will be added later.
    b. If deploying via Prism Central, the disk size can be altered during deployment for 4, 8, or 16 core vSensors.
    c. DO NOT Power on the vSensor VM after creation because additional configuration changes must be made before booting the vSensor.
2. Increase vSensor Disk Size: If deploying via PE, and you are deploying a 4, 8, or 16 core vSensor, increase the disk size of the vSensor using the ACLI.
3. Create network function provider category and value using the Prism Central UI.
4. Using Prism Element, retrieve cluster details – we'll need the name, UUID, and CVM IP for configuring the network function chain and accessing the ACLI.
5. Create a network function chain in your AHV cluster using PC API.
6. Update the powered off vSensor VM using both the ACLI and PC GUI or PC API.
    a. Update the powered off vSensor VM using ACLI on your CVM by performing the following:
        i. Set the VM to be an agent and system VM.
        ii. Create the network function NIC (this will be the capture port of the vSensor) and set it to TAP type.
        iii. Set VM affinity to the specific AHV host you intend to use it on (can also be done in PE GUI).
        iv. Get the UUID of the network function VM (the vSensor you are deploying).
    b. Add the network function provider category and value to the vSensor VM using Prism Central UI.
7. Direct traffic to the configured network function chain.
    a. Repeat step 7 to add additional subnets as often as required.

When deploying multiple vSensors or deploying in multiple AHV clusters, keep in mind the following:
- A single network function chain is required for each AHV cluster in your Nutanix deployment.
- A Vectra vSensor is required for each node of your AHV cluster when using Service Chaining 1.0.
    - The vSensor should be sized to handle the expected traffic volume from the subnets that will be observed on each respective node of the cluster.

1. Create vSensor VM - DO NOT POWER ON AFTER CREATION

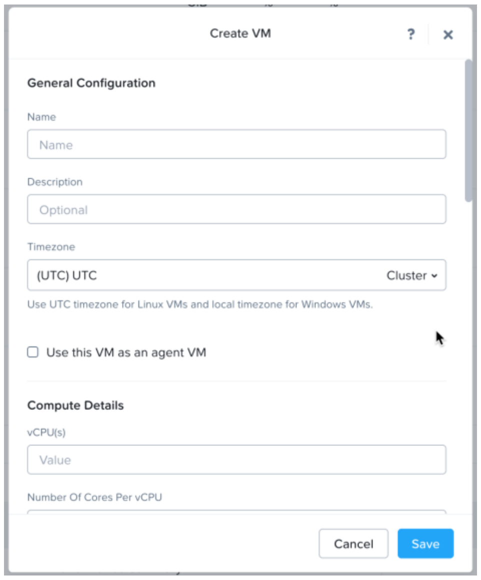
Next we will create a vSensor VM using either Prism Element (if you uploaded the vSensor images using PE) or Prism Central (if you uploaded the vSensor using PC).

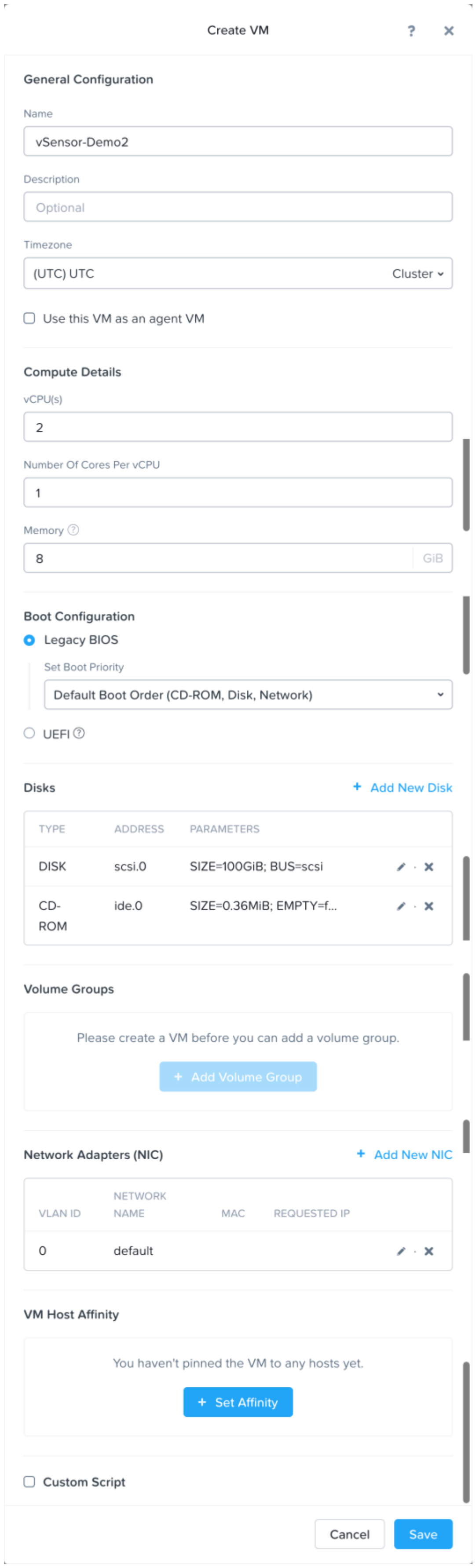
Please Note: DO NOT POWER ON THE VSENSOR AFTER CREATION

- Additional configuration changes must be made before booting the vSensor.

Prism Element (PE) vSensor Creation Notes

If your vSensor images were uploaded to PE, navigate in PE to *VM* > *"+ Create VM"* and a scrolling dialog box similar to the below will open where you will need to fill out the required details for the VM. On the next page an example of a fully configured dialog box along with guidance for each choice.





Create VM

General Configuration

Name: vSensor-Demo2

Description: Optional

Timezone: (UTC) UTC — Cluster

☐ Use this VM as an agent VM

Compute Details

vCPU(s): 2

Number Of Cores Per vCPU: 1

Memory: 8 GiB

Boot Configuration

● Legacy BIOS

Set Boot Priority: Default Boot Order (CD-ROM, Disk, Network)

○ UEFI

Disks — + Add New Disk

TYPE	ADDRESS	PARAMETERS
DISK	scsi.0	SIZE=100GiB; BUS=scsi
CD-ROM	ide.0	SIZE=0.36MiB; EMPTY=f...

Volume Groups

Please create a VM before you can add a volume group.

+ Add Volume Group

Network Adapters (NIC) — + Add New NIC

VLAN ID	NETWORK NAME	MAC	REQUESTED IP
0	default		

VM Host Affinity

You haven't pinned the VM to any hosts yet.

+ Set Affinity

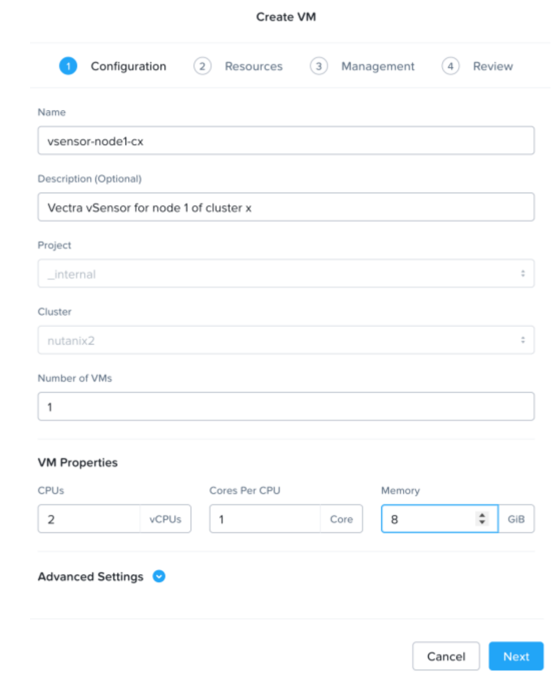
☐ Custom Script

Cancel Save

- **Name** – Use a name of your choice.
- **Description** – Optional.
- **Timezone** – Configure per your standards.
- **Use this VM as an agent VM** – We will enable this at the ACLI later or you can enable this now.
- **vCPU(s)** – 2, 4, 8, or 16 depending on the chosen configuration.
  - Look at Nutanix vSensor Requirements and Throughput for guidance.
- **Number of Cores Per vCPU** – Always enter 1.
- **Memory** – 8, 8, 16, or 64 GB depending on your chosen configuration.
- **Boot Configuration** – Legacy BIOS and leave the "Set Boot Priority" at its default.
- **Disks**
  - Delete the existing CD-ROM.
  - Add the QCOW2 image as a new disk.
    - **Type** – Disk.
    - **Operation** – Clone from Image Service.
    - **Bus Type** – SCSI.
    - **Image** – Select the QCOW2 image that you uploaded.
    - **Size** – This will be greyed out. If this needs to be modified based on your configuration, we will do this using the ACLI later.
    - **Index** – Next Available.
  - Add the seed.iso image as a new CD-ROM.
    - **Type** – CD-ROM.
    - **Operation** – Clone from Image Service.
    - **Bus Type** – IDE.
    - **Image** – Select the seed.iso image that you uploaded.
    - **Size** – This will be greyed out and does not need to be modified.
    - **Index** – Next Available.
- **Volume Groups** – Ignore.
- **Network Adapters** – Add a NIC on the network of your choosing where you want the management (MGT1) interface of the vSensor to attach.
- **VM Host Affinity** – We will set this later at the ACLI or you can choose the node you wish to PIN to now. A vSensor should be installed on each node in the AHV cluster.
- **Custom Script** – Ignore.
- When done click the "Save" button but do NOT power on your VM.

Prism Central (PC) vSensor Creation Notes

If your vSensor images were uploaded to PC, navigate in PC to *Infrastructure* > *Compute* > *VMs* > *Create VM* and you will be presented with a four page dialog to create the vSensor VM:

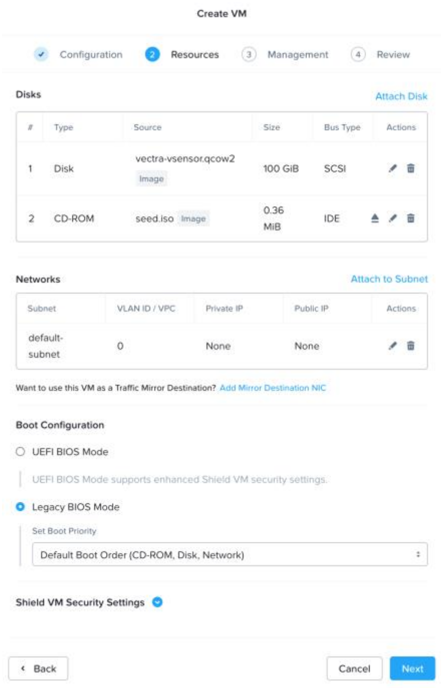


Name	vsensor-node1-cx
Description (Optional)	Vectra vSensor for node 1 of cluster x
Project	_internal
Cluster	nutanix2
Number of VMs	1

VM Properties

CPUs	Cores Per CPU	Memory
2 vCPUs	1 Core	8 GiB

Advanced Settings



Disks

#	Type	Source	Size	Bus Type	Actions
1	Disk	vectra-vsensor.qcow2 Image	100 GiB	SCSI	
2	CD-ROM	seed.iso Image	0.36 MiB	IDE	

Networks

Subnet	VLAN ID / VPC	Private IP	Public IP	Actions
default-subnet	0	None	None	

Want to use this VM as a Traffic Mirror Destination? Add Mirror Destination NIC

Boot Configuration

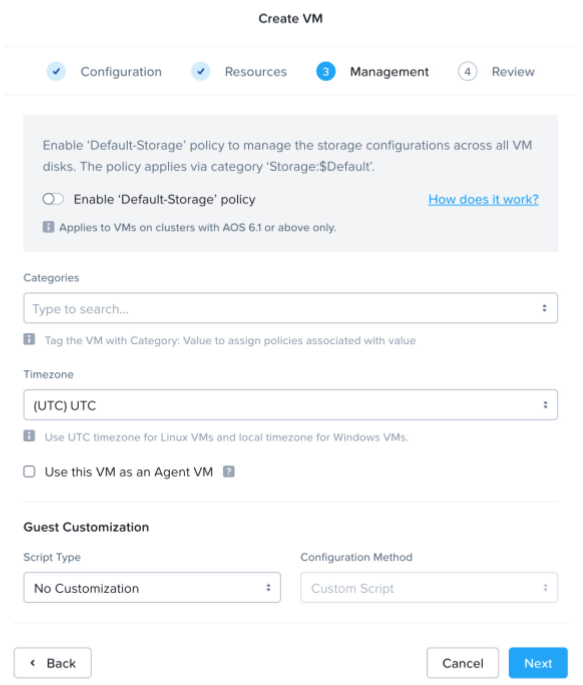
○ UEFI BIOS Mode

UEFI BIOS Mode supports enhanced Shield VM security settings.

● Legacy BIOS Mode

Set Boot Priority: Default Boot Order (CD-ROM, Disk, Network)

Shield VM Security Settings



Enable 'Default-Storage' policy to manage the storage configurations across all VM disks. The policy applies via category 'Storage:$Default'.

Enable 'Default-Storage' policy

Applies to VMs on clusters with AOS 6.1 or above only.

Categories: Type to search...

Tag the VM with Category: Value to assign policies associated with value

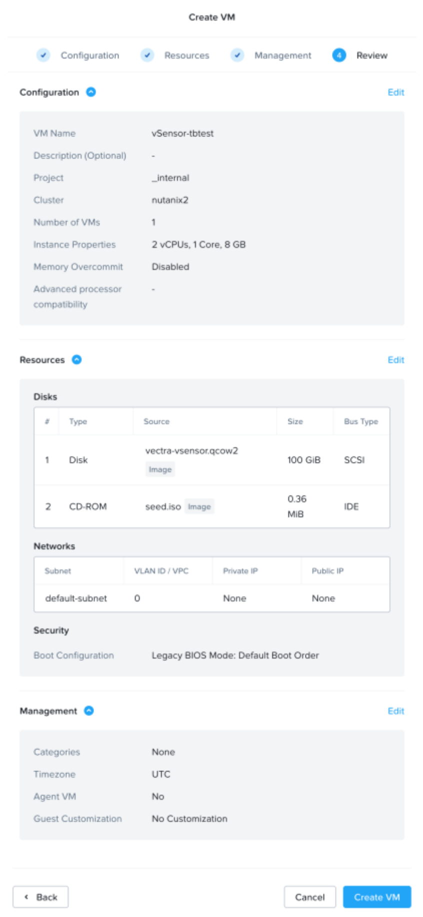
Timezone: (UTC) UTC

Use UTC timezone for Linux VMs and local timezone for Windows VMs.

☐ Use this VM as an Agent VM

Guest Customization

Script Type	Configuration Method
No Customization	Custom Script



Configuration

VM Name	vSensor-tbtest
Description (Optional)	-
Project	_internal
Cluster	nutanix2
Number of VMs	1
Instance Properties	2 vCPUs, 1 Core, 8 GB
Memory Overcommit	Disabled
Advanced processor compatibility	-

Resources

Disks

#	Type	Source	Size	Bus Type
1	Disk	vectra-vsensor.qcow2 Image	100 GiB	SCSI
2	CD-ROM	seed.iso Image	0.36 MiB	IDE

Networks

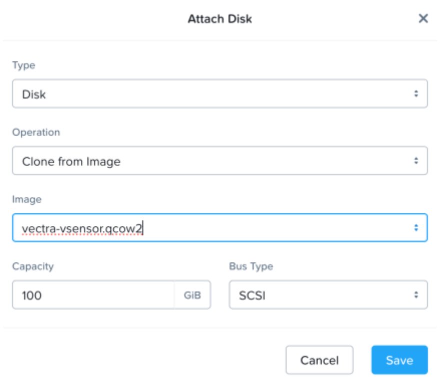
Subnet	VLAN ID / VPC	Private IP	Public IP
default-subnet	0	None	None

Security

Boot Configuration	Legacy BIOS Mode: Default Boot Order

Management

Categories	None
Timezone	UTC
Agent VM	No
Guest Customization	No Customization



Attach Disk

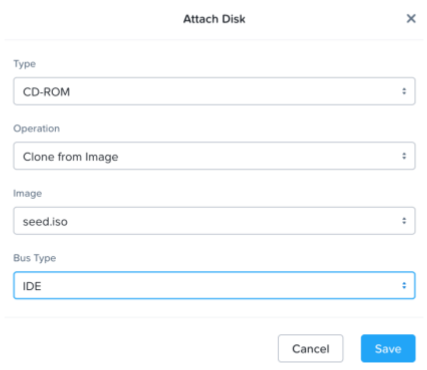
Type: Disk

Operation: Clone from Image

Image: vectra-vsensor.qcow2

Capacity: 100 GiB

Bus Type: SCSI



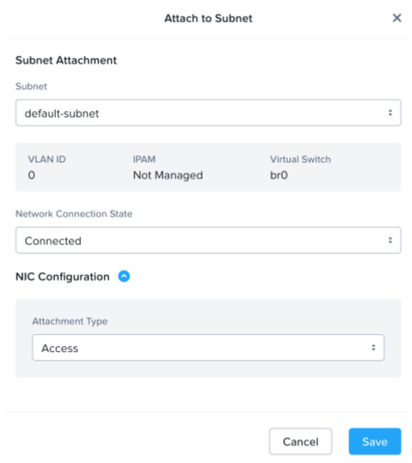
Attach Disk

Type: CD-ROM

Operation: Clone from Image

Image: seed.iso

Bus Type: IDE



Attach to Subnet

Subnet Attachment

Subnet: default-subnet

VLAN ID	IPAM	Virtual Switch
0	Not Managed	br0

Network Connection State: Connected

NIC Configuration

Attachment Type: Access

Notes on Prism Central vSensor VM Creation:

- Page 1 “Configuration”
  - **Name** – Use a name of your choice.
  - **Description** – Optional.
  - **Project/Cluster** – Set if required for your org.
  - **Number of VMs**
    - Normally 1 if deploying a single vSensor.
    - This can be set to more than 1 if you are deploying multiple identically configured vSensors. The additional vSensors would be named with -1, -2, etc after the configured name.
  - **VM Properties**
    - Refer to Nutanix vSensor Requirements and Throughput for vSensor specification.
    - **CPUs** – Set to 2, 4, 8, or 16 based on the size vSensor you are deploying.
    - **Cores** – Always set to 1.
    - **Memory** – Set to 8, 8, 16, or 64 based on the size of vSensor you are deploying.
  - **Advanced Settings** – Not required.
- Page 2 “Resources”
  - **Disks** – Add the disks per the screenshots above, cloning them from the uploaded vSensor images.
    - Disk size can be changed per the vSensor specifications to avoid doing it in Step 2 below.
  - **Network** – Attach to the subnet of your choice.
    - This will be where the Mangement (MGT1) interface of the vSensor will be connected.
    - Initially it will boot with DHCP, but this can be changed to static at the CLI of the vSensor.
      - See 11. Initial vSensor configuration at the Vectra vSensor CLI for details.
  - **Boot Configuration**
    - This must be set to “Legacy BIOS Mode” with “Default Boot Order (CD-ROM, Disk, Network).
    - The seed.iso (CD-ROM) is read during vSensor boot to help configure network adapters.
  - **Shield VM Security Settings** – Not required.

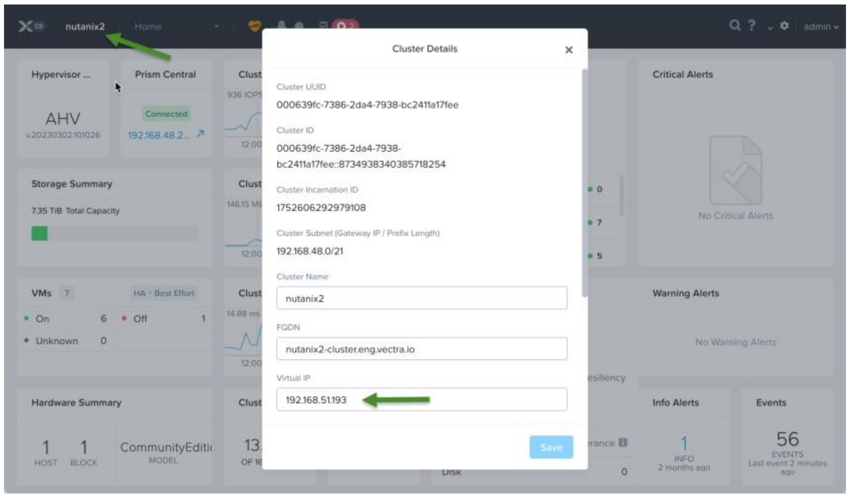
© 2025 Vectra AI, Inc.

- Page 3 “Management”
    - **Enable ‘Default-Storage’ Policy**
        - Whether you enable this or not will depend on your defaults.
        - As per the Nutanix vSensor Requirements and Throughput:
            - Vectra recommends that Sensors are configured to use storage local to the hypervisor and are not stored on a SAN. Vectra vSensors require extremely high throughput from their disk storage and this throughput cannot normally be sustained by SAN systems without impact to other SAN users.
        - Also keep in mind that you will be pinning (setting affinity) for each vSensor to a specific node in each cluster.
    - **Categories**
        - If you don’t already have a category and value from prior vSensor deployments, leave this blank for now.
        - If you have already created the category and value in 5. Create network function provider category and value and are deploying a new vSensor in this cluster, you can set the “network_function_provider” category and value that you previously created and won’t have to do it later in 8. Update the Powered off VM Using Both the ACLI and Prism Central UI.
    - **Timezone** – Configure per your standards.
    - **Use this VM as an Agent VM**
        - The vSensor needs to be set as an Agent VM.
        - This can be enabled now, or it can be set during 8. Update the Powered off VM Using Both the ACLI and Prism Central UI. Some configuration must be done at the CVM ACLI later anyway.
    - **Guest Customization** – Not required.
- Page 4 “Review”
    - Review the settings, go back and change anything if required, and when ready, click “Create VM”.

2. Increase vDisk Size of Vectra vSensor VM (if required)

If you didn’t increase the disk size during the vSensor VM creation (only possible when creating via Prism Central), do it now. This is only required for VMs requiring a disk larger than the default of 100GB. Replace <vectra-sensor-VM-name> with the name of your Vectra Sensor VM. See Nutanix vSensor Requirements and Throughput for sizes.

Login to one of your controller VMs (CVM), and enter the ACLI command line. You can use the CVM Virtual IP from Prism Element. To find this, click on your cluster name in Prism Element and copy the “Virtual IP”.



Cluster Details

Cluster UUID
000639fc-7386-2da4-7938-bc2411a17fee

Cluster ID
000639fc-7386-2da4-7938-bc2411a17fee::8734938340385718254

Cluster Incarnation ID
1752606292979108

Cluster Subnet (Gateway IP / Prefix Length)
192.168.48.0/21

Cluster Name
nutanix2

FQDN
nutanix2-cluster.eng.vectra.io

Virtual IP
192.168.51.193

```
acli vm.list
This will show you the list of VMs and allow you to easily see the VM name if you don’t have it handy.
acli vm.get <vectra-sensor-VM-name>
```

Find the disk in the output and note the device_uuid as shown highlighted below (The full output is much longer we are showing only the disk_list portion that is relevant):

```
disk_list {
      addr {
        bus: "scsi"
        index: 0
      }
      container_id: 4
      container_uuid: "4adfaee8-5d13-43d0-9e2c-0acb78c20bf3"
      device_uuid: "85c6d790-7aec-4976-b8f7-a399b69f7597"
      naa_id: "naa.6506b8d2082195bbdbc2a81ffab4b6f0"
      vmdisk_size: 26843545600
      vmdisk_uuid: "5d5d00bb-3fa7-4a06-acb8-ba95f4efdab3"
    }
```

Use this device_uuid to expand the disk size to match the storage requirements.

Syntax:

```
acli vm.disk_update <vectra-sensor-VM-name> device_uuid=<device-uuid> new_size=<new disk size>
```

Example:

```
acli vm.disk_update <vectra-sensor-VM-name> device_uuid=85c6d790-7aec-4976-b8f7-a399b69f7597 new_size=150G
```

3. Create Network Function Provider Category and Value

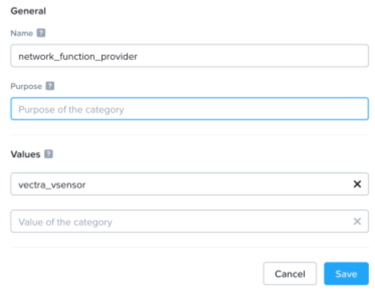
All of the vSensor VMs need to be configured with a category called “network_function_provider” with a value of your choice. Vectra suggests using “vectra_vsensor” for the value, but this can be any value that you will consistently use moving forward. vSensor Deployment Step 3 Alternate API Instructions can be used if you wish to do this via API.

In this step, you will create the category and value.

- A category with a key of “network_function_provider” will be created in Prism Central.
- The values of this key will act as labels on the network function VMs that identify them on each host.
- The key name of “network_function_provider” must not be changed.
- Later, when the network function chain (NFC) is configured, it will also be tied to this category and value which means that the NFC will apply to any Vectra vSensor VM that has the same category and value.

Instructions:

- Navigate in your Prism Central (PC) UI to *Infrastructure > Administration > Categories* and click “New Category”.
    - **Name** – This must be set to “network_function_provider” and cannot be altered.
    - **Purpose** – Can be left blank or enter the text of your choice.
    - **Values** – Vectra suggests “vectra_vsensor” but this can be any value that you will use consistently.



General

Name: network_function_provider

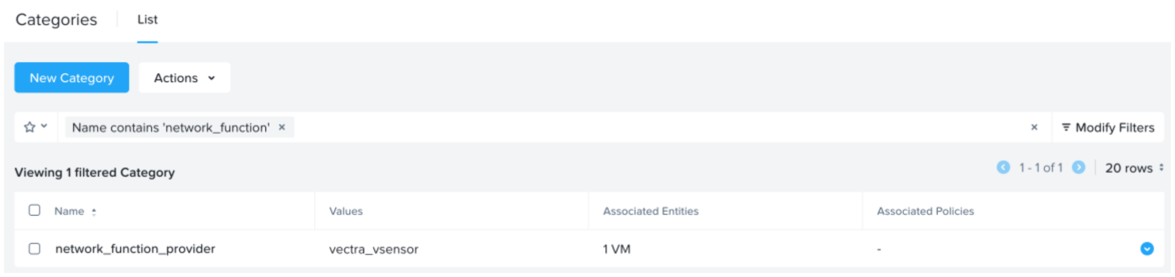
Purpose: Purpose of the category

Values

vectra_vsensor

Value of the category

- When you type in a value, a new box appears. You do not need to do anything with this. Just "Save" when you are done.
- After saving, search for your newly created category in the category list and validate it was created properly.



Categories | List

New Category | Actions

Name contains 'network_function' | Modify Filters

Viewing 1 filtered Category — 1 - 1 of 1 — 20 rows

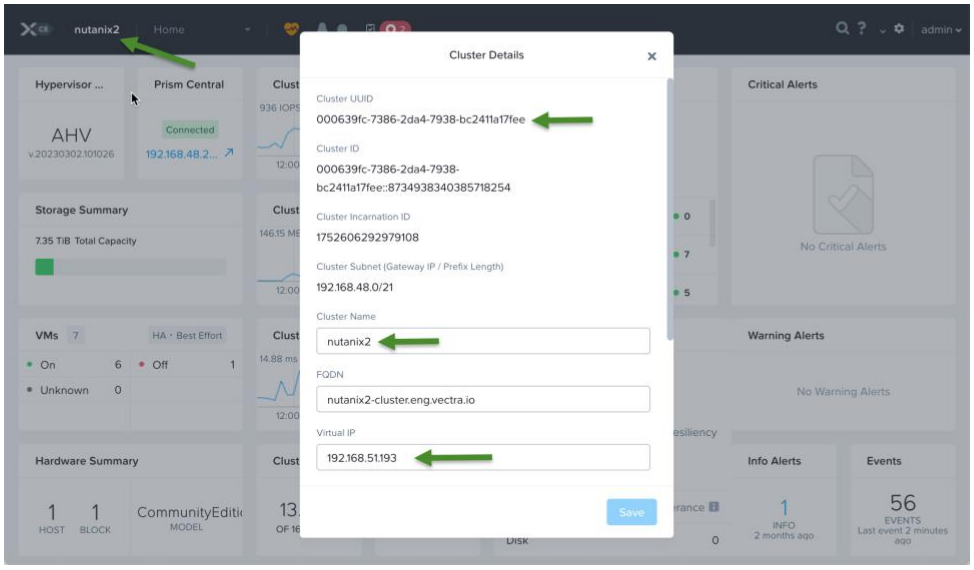
Name	Values	Associated Entities	Associated Policies
network_function_provider	vectra_vsensor	1 VM	-

4. Retrieve Cluster Name / UUID Where the Network Function Chain Will Be Created

vSensor Deployment Step 4 Alternate API Instructions can be used if you wish to do this via API. It is recommended to do this step via the UI instructions below as they are typically easier for customers.

At this point it is useful to make sure that you are keeping track of data that you will need to recall in later parts of the overall deployment process. Please make sure that you keep track of the following:

- From the cluster details page in Prism Element. Click on your cluster name in Prism Element



Cluster Details

Cluster UUID: 000639fc-7386-2da4-7938-bc2411a17fee

Cluster ID: 000639fc-7386-2da4-7938-bc2411a17fee::8734938340385718254

Cluster Incarnation ID: 1752606292979108

Cluster Subnet (Gateway IP / Prefix Length): 192.168.48.0/21

Cluster Name: nutanix2

FQDN: nutanix2-cluster.eng.vectra.io

Virtual IP: 192.168.51.193

  - **Network Function Provider**
    - The value you created in step 5. Vectra recommends "vectra_vsensor"
  - **CVM Virtual IP**
    - Any CVM (there is one per node) in the cluster can be used, the virtual IP is usually easiest.
    - Some configuration steps must be done using the ACLI later.
  - **Cluster Name** – Will be required when creating the network function chain in the next step.
  - **Cluster UUID** - Will be required when creating the network function chain in the next step.
- In Step 5.2 you will also collect:
  - **Network Function Chain UUID** – UUID for later use when assigning subnets to the NFC.
    - You want the UUID from the metadata portion of the response.

5. Create the Network Function Chain in the AHV Cluster

This step requires Prism Central API calls. You will create a network function chain (NFC) in the AHV cluster using the network_function_provider value, cluster name, and UUID gathered in the previous steps. You will also provide a "name" value for the NFC. In our example below our network function chain will have a value of "vectra_tap".

- Please note that if you are deploying in multiple AHV clusters, you will need to repeat network function chain setup for each cluster you are deploying Vectra vSensors in.

Step 1: Create the network function chain with a name of your choice using the gathered values.

```
POST - https://{{pc_ip}}:9440/api/nutanix/v3/network_function_chains
```

Body:

- Modify the body with the name you wish to use for the network function chain (we suggest "vectra_tap", the network_function_provider category value we created earlier (we are using "vectra_vsensor" and the name and UUID of the cluster you are deploying this NFC in.

```
{
    "spec": {
        "name": "vectra_tap",
        "resources": {
            "network_function_list": [
                {
                    "network_function_type": "TAP",
                    "category_filter": {
                        "type": "CATEGORIES_MATCH_ANY",
                        "params": {"network_function_provider": ["vectra_vsensor"]}
                    }
                }
            ]
        },
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2 ",
            "uuid": "0005dd45-7584-73d2-07af-52540083d046"
        }
    },
    "api_version": "3.1.0",
    "metadata": {
        "kind": "network_function_chain"
    }
}
```

Response example:

```
{
    "status": {
        "state": "PENDING",
        "execution_context": {
            "task_uuid": "3acead7a-bfec-4adb-b75c-733f191631ef"
        }
    },
    "spec": {
        "name": "vectra_tap",
        "resources": {
            "network_function_list": [
                {
                    "network_function_type": "TAP",
                    "category_filter": {
                        "params": {
                            "network_function_provider": [
                                "vectra_vsensor"
                            ]
                        },
                        "type": "CATEGORIES_MATCH_ANY"
                    }
                }
            ]
        },
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
        }
    },
    "api_version": "3.1",
    "metadata": {
        "owner_reference": {
            "kind": "user",
            "uuid": "00000000-0000-0000-0000-000000000000",
            "name": "admin"
        },
        "use_categories_mapping": false,
        "kind": "network_function_chain",
        "spec_version": 0,
        "uuid": "24936fa3-54e5-4ae2-a80c-411fae635236"
    }
}
```

Step 2: Verify that the network function chain is created.

- Using earlier guidance, you can check the task UUID for completion if desired.
- In this step, we will ask the API to list the network function chains to validate creation.
- The UUID of the network function chain that is listed in the metadata portion of the response example below will need to be used Step 7: Direct traffic to the configured network function chain. Please keep track of this.

```
POST - https://{{pc_ip}}:9440/api/nutanix/v3/network_function_chains/list
```

Body:

```
{
"kind": "network_function_chain"
}
```

Abbreviate Response Example:

```
{
    "api_version": "3.1",
    "metadata": {
        "kind": "network_function_chain",
        "offset": 0,
        "length": 1,
        "total_matches": 1
    },
    "entities": [
        {
            "status": {
                "name": "vectra_tap",
                "resources": {
                    "network_function_list": [
                        {
                            "network_function_type": "TAP",
                            "category_filter": {
                                "type": "CATEGORIES_MATCH_ALL",
                                "params": {
                                        "network_function_provider": [
                                            "vectra_vsensor"
                                        ]
                                    }
                                }
                            }
                        ]
                    },
                    "state": "COMPLETE",
                    "execution_context": {
                        "task_uuid": [
                            "654f65d7-4e19-4222-8770-73b88129ba26"
                        ]
                    }
                },
                "metadata": {
                    "uuid": "9385b9d5-d2cf-4039-b3b1-45f5ef744e1a",
                    "spec_hash": "00000000000000000000000000000000000000000000000000",
                    "spec_version": 0,
                    "owner_reference": {
                        "kind": "user",
                        "uuid": "71f50668-79f9-50d4-8dec-eaaf5a411f36",
                        "name": "admin"
                    },
                    "categories": {},
                    "categories_mapping": {},
                    "creation_time": "2025-09-15T20:25:10Z",
                    "last_update_time": "2025-09-15T20:25:11Z",
                    "kind": "network_function_chain"
                },
                "spec": {
                    "cluster_reference": {
                        "kind": "cluster",
                        "name": "nutanix2",
                        "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
                    },
                    "name": "vectra_tap",
                    "resources": {
                        "network_function_list": [
                            {
                                "category_filter": {
                                    "params": {
                                        "network_function_provider": [
                                            "vectra_vsensor"
                                        ]
                                    },
                                    "type": "CATEGORIES_MATCH_ANY"
                                },
                                "network_function_type": "TAP"
                            }
                        ]
                    }
                }
            }
        ]
}
```

6. Update the Powered off VM Using Both the ACLI and Prism Central UI

6.1 Overview:

- Set the VM to be an agent and system VM.
- Create the network function NIC and set it to TAP type.
- Tie the vSensor VM to a specific node so that is not moved from this node and captures traffic on this node.
    - This is setting an affinity to a specific node (also known as pinning).
    - If you already did this during Prism Element (PE) vSensor Creation, you do not need to do this again.
    - If you created the VM using Prism Central, you must set the affinity using the ACLI or the Prism element UI.

6.2 Overview:

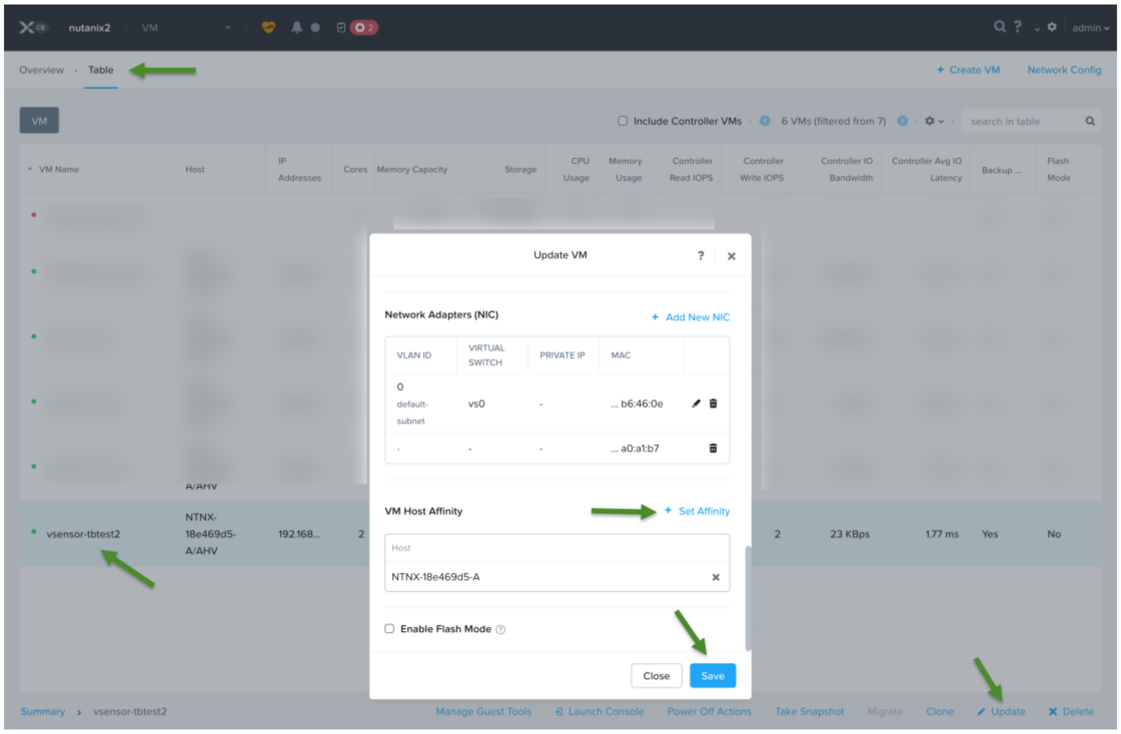
- Add the network function provider category and value to the still powered off vSensor.

6.1 - Update the Network Function VM (vSensor)

- Log in to a CVM in your target cluster and access the ACLI.
- Execute the below ACLI commands for each network function VM (vSensor).
    - One vSensor is needed for every node and assign affinity to each node.
- You may have already set the vSensor as an agent VM during creation, if so, you don't need to do it again, but there is no harm in just copying/modifying the entire command below and doing it again.
- vSensor_Name should be replaced by your vSensor name and x.x.x.x should be the IP of the node you are setting affinity to.

Affinity can also be set in the GUI without needing to find the IP of the node you are setting affinity to:

- Navigate in your Prism Element UI (for the cluster you are working on) to the VM table view, select your vSensor VM and click the "Update" button.
- Scroll down to the bottom and set the affinity to the specific node you desire.



VM Name	Host	IP Addresses	Cores	...	Controller Write IOPS	Controller IO Bandwidth	Controller Avg IO Latency	Backup ...	Flash Mode
vsensor-tbtest2	NTNX-18e469d5-A/AHV	192.168...	2	...	2	23 KBps	1.77 ms	Yes	No

VLAN ID	VIRTUAL SWITCH	PRIVATE IP	MAC
0 default-subnet	vs0	-	... b6:46:0e
-	-	-	... a0:a1:b7

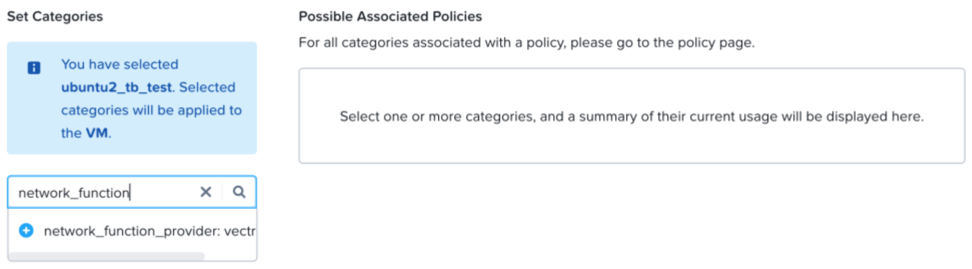
Host
NTNX-18e469d5-A

```
nutanix@CVM $ acli
<acropolis> vm.update vSensor_Name agent_vm=true extra_flags=is_system_vm=true
<acropolis> vm.nic_create vSensor_Name type=kNetworkFunctionNic network_function_nic_type=kTap
<acropolis> vm.affinity_set vSensor_Name host_list=X.X.X.X
```

6.2 - Add the Network Function Provider Category/Value to the vSensor

vSensor Deployment Step 6 Alternate API Instructions can be used if you wish to do this via API. It is recommended to do this step via the UI instructions below as they are typically easier for customers.

- Navigate in your Prism Central UI to *Infrastructure* > *Compute, VMs*.
- Right click on your vSensor VM and choose *Other Actions* > *Manage Categories.*
- Search for “network_function” and find the network_function_provider category and value you created earlier.
    - The display can be scrolled inside the box to see the entire value for the category.
- Save the setting in the bottom right.



Set Categories

You have selected **ubuntu2_tb_test**. Selected categories will be applied to the **VM**.

network_function

network_function_provider: vectr

Possible Associated Policies

For all categories associated with a policy, please go to the policy page.

Select one or more categories, and a summary of their current usage will be displayed here.

7. Direct Traffic to the Configured Network Function Chain

Nutanix supports 3 ways to direct traffic to the network function chain.

- Flow security policies to selectively direct specific traffic flows to the network function chain.
    - This requires Flow licenses and is not in scope for this guide.
- Directing an entire AHV network to the network function chain.
    - This is the example we will cover below.
    - Directing an entire AHV network through the network function chain does require Prism Central but does NOT require Flow licenses or microsegmentation.
    - An AHV network is called a "subnet" in the Prism Central v3 API.
    - This is the most likely scenario we expect customers to encounter. It allows for all the virtual traffic on an AHV network on a single hypervisor to be copied to the vSensor to allow for inspection.
    - The steps outlined in Step 9 should be repeated for each subnet you want to direct to the vSensor.
        - Nothing needs to change with the vSensor configuration when you add or remove subnets to the network function chain. The vSensor does not need to be rebooted or told to analyze different subnets. The vSensor simply produces metadata for any traffic that it receives from the network function chain.
- Direct a single VM NIC to the network function chain.
    - This is not in scope for this guide.

For options not covered in this guide, please work with your Nutanix account team to get assistance if you are not already familiar with the configuration of Flow or directing a single VM NIC to the network function chain.

- As stated in the Traffic Capture Options, only VLAN Basic subnets are supported, if you are using NCVLAN Subnets, please note that NCVLAN does not support network function chains.
    - Traffic Mirroring supports functioning in environments that use NCVLAN Subnets.

Overview of Required Steps

7.1: List Subnets

- In this step you will list the subnets and choose which subnets you wish to be routed through the network function chain.

7.2: Add Network Function Chain (NFC) Reference to Subnet

- In this step you will update the subnet to reference the NFC.

7.3: Validate NFC is Referenced in Subnet Details

- In this step you will an API call to validate that the subnet now references the NFC.

7.4: (Optional) – Remove NFC Reference From a Subnet

- This step is optional and provided for an easy way to clear an NFC reference from a subnet. NFCs cannot be deleted when a subnet still references them. In some environments, "extra" NFCs may exist from prior attempts to configure service chaining in Nutanix.

7.1 List Subnets

- Access the ACLI of a CVM in your cluster (see 6. Retrieve Cluster Name / UUID Where the Network Function Chain Will Be Created for how to find the CVM IP if you don't still have it handy.)
- Use the "**`net.list`**" command to show the subnets in your cluster.

```
<acropolis> net.list
Network name    Network UUID                          Type      Identifier  Virtual Switch  Subnet
default-subnet  f5d8c6bb-8ad9-42c5-b977-8bafbba10b81  kBridged  0           vs0
test            5a4a32d9-76f6-4b40-8eda-eb2fa46899bc  kBridged  100         vs0
```

- In our example above, we have two subnets. You will need the Network UUIDs of the subnets that you want to be mapped to the NFC.

7.2 Add Network Function Chain (NFC) Reference to Subnet

- Use the "net.update_network_function_chain" to update each desired subnet to reference the NFC.

```
net.update_network_function_chain <Network UUID> <NFC UUID>
net.update_network_function_chain f5d8c6bb-8ad9-42c5-b977-8bafbba10b81 472f2d96-472d-4e6e-94d1-1ea66f2be89d
```

- In the above example, we are adding a reference for the NFC UUID beginning with 472f2d96 to the default subnet UUID we found in step 9.1. Repeat this step for each subnet you wish to map to the NFC.

7.3 Validating the Subnet is Updated to Reference the NFC

- Use an empty body in your request and put the desired subnet you want details for in the request URL.

```
https://{{pc_ip}}:9440/api/nutanix/v3/subnets/f5d8c6bb-8ad9-42c5-b977-8bafbba10b81
```

Response Example:

```
{
    "api_version": "3.1",
    "metadata": {
        "last_update_time": "2025-09-24T15:04:50Z",
        "kind": "subnet",
        "uuid": "f5d8c6bb-8ad9-42c5-b977-8bafbba10b81",
        "spec_version": 8,
        "creation_time": "2025-09-10T20:39:29Z",
        "categories_mapping": {},
        "categories": {},
        "owner_reference": {
            "kind": "user",
            "uuid": "71f50668-79f9-50d4-8dec-eaaf5a411f36",
            "name": "tbilen"
        },
        "spec_hash": "00000000000000000000000000000000000000000000000000"
    },
    "spec": {
        "resources": {
            "subnet_type": "VLAN",
            "vswitch_name": "br0",
            "network_function_chain_reference": {
                "kind": "network_function_chain",
                "uuid": "472f2d96-472d-4e6e-94d1-1ea66f2be89d"
            },
            "vlan_id": 0,
            "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755"
        },
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
        },
        "name": "default-subnet"
    },
    "status": {
        "name": "default-subnet",
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
        },
        "resources": {
            "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755",
            "vswitch_name": "br0",
            "network_function_chain_reference": {
                "kind": "network_function_chain",
                "uuid": "472f2d96-472d-4e6e-94d1-1ea66f2be89d"
            },
            "vlan_id": 0,
            "subnet_type": "VLAN",
            "ip_usage_stats": {
                "num_macs": 8
            }
        },
        "state": "COMPLETE",
        "execution_context": {
            "task_uuid": [
                "757c3ee0-decc-4638-9491-ab4ac8c292a7"
            ]
        }
    }
}
```

- As you can see in the above example, the network function chain reference appears in the response.

7.4 Remove NFC Reference From a Subnet (or Delete an NFC) (Optional)

If you ever need to delete a NFC or remove an NFC reference from a subnet, you can use the commands in this section to remove the NFC reference from a subnet. NFCs cannot be deleted when a subnet still references them. In some environments, “extra” NFCs may exist from prior attempts to configure service chaining in Nutanix.

- To remove NFC reference from a subnet use the “**`net.clear_network_function_chain`**” command from your CVM ACLI as shown below:

```
net.clear_network_function_chain <network UUID>
```

- You can use the API call detailed in 7. Create the Network Function Chain in the AHV Cluster to list the NFCs in your cluster.
- You can use a “**`DELETE`**” call to the **`/api/nutanix/v3/network_function_chains/<NFC_UUID>`** Prism Central API endpoint to delete a NFC. Replace **`<NFC_UUID>`** with the UUID of the NFC you wish to delete.

Body Example:

```
{
  "kind": "network_function_chain"
}
```

Nutanix vSensor Deployment for Traffic Mirroring

Traffic Mirroring Deployment Overview

Prism Central is the only supported method for deployment using Traffic Mirroring as a steering method. Prism Central is required to create and manage Traffic Mirroring Sessions in the UI. Traffic Mirroring Sessions can also be created in the ACLI, but this is not in scope for this guide and Nutanix has stated to Vectra that ACLI access is planned to be removed in a future release and therefore, Vectra is not documenting CLI based Traffic Mirroring.

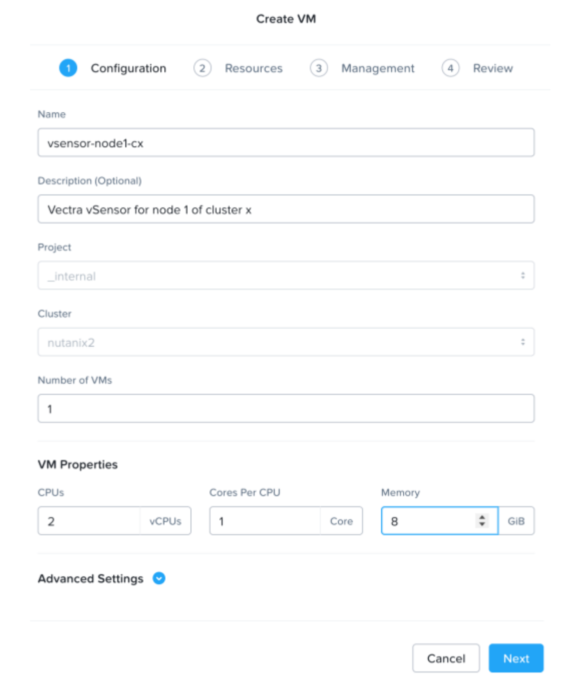
Below is an overview of the major steps:

1. Create vSensor VM
    a. One NIC will a standard NIC that will be used for management (MGT1).
    b. A special "Mirror Destination NIC" will be added to be used as the vSensor capture port. Only Mirror Destination NICs can be used as destinations in a traffic mirroring session.
    c. The disk size can be altered during deployment for 4, 8, or 16 core vSensors.
        i. 2 core vSensors do not need their disk size altered.
        ii. If you miss this step, with the vSensor VM powered off, you can follow the instructions from 2. Increase vDisk Size of Vectra vSensor VM (if required) to alter disk size after deployment or change the size by updating the VM in Prism Central UI.
2. Set VM affinity to the specific AHV host you intend to use it on (can also be done in PE GUI).
3. Perform Shared Steps After vSensor Deployment
4. Create a Traffic Mirroring Session
    a. Add additional vSensors and create/update traffic mirroring sessions as required for your deployment.

© 2025 Vectra AI, Inc.

1. Create vSensor VM in Prism Central

Navigate in Prism Central to *Infrastructure* > *Compute* > *VMs* > *Create VM* and you will be presented with a four-page dialog to create the vSensor VM:



Name
vsensor-node1-cx

Description (Optional)
Vectra vSensor for node 1 of cluster x

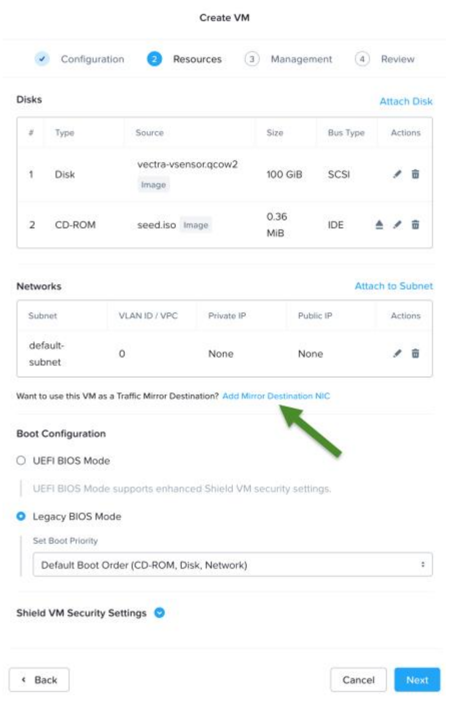
Project
_internal

Cluster
nutanix2

Number of VMs
1

VM Properties

CPUs	Cores Per CPU	Memory
2 vCPUs	1 Core	8 GiB



Disks

#	Type	Source	Size	Bus Type	Actions
1	Disk	vectra-vsensor.qcow2 Image	100 GiB	SCSI	
2	CD-ROM	seed.iso Image	0.36 MiB	IDE	

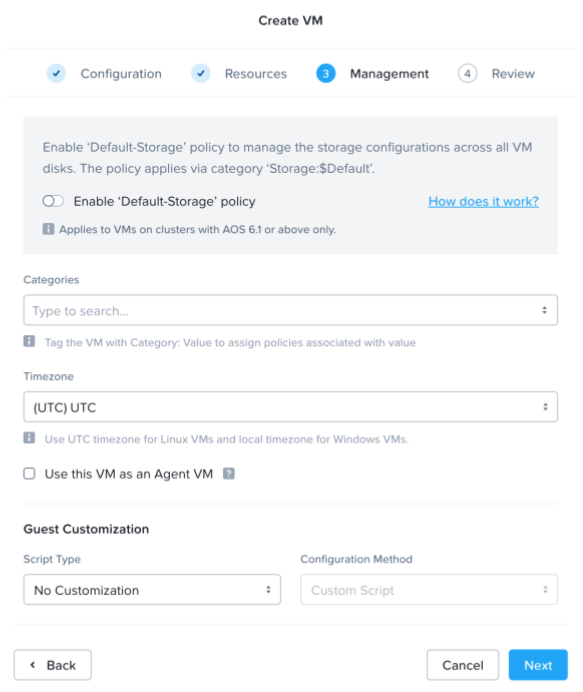
Networks

Subnet	VLAN ID / VPC	Private IP	Public IP	Actions
default-subnet	0	None	None	

Want to use this VM as a Traffic Mirror Destination? Add Mirror Destination NIC

Boot Configuration

UEFI BIOS Mode
UEFI BIOS Mode supports enhanced Shield VM security settings.
[✓ Legacy BIOS Mode]
Set Boot Priority
Default Boot Order (CD-ROM, Disk, Network)



Enable 'Default-Storage' policy to manage the storage configurations across all VM disks. The policy applies via category 'Storage:$Default'.

Enable 'Default-Storage' policy   How does it work?

Applies to VMs on clusters with AOS 6.1 or above only.

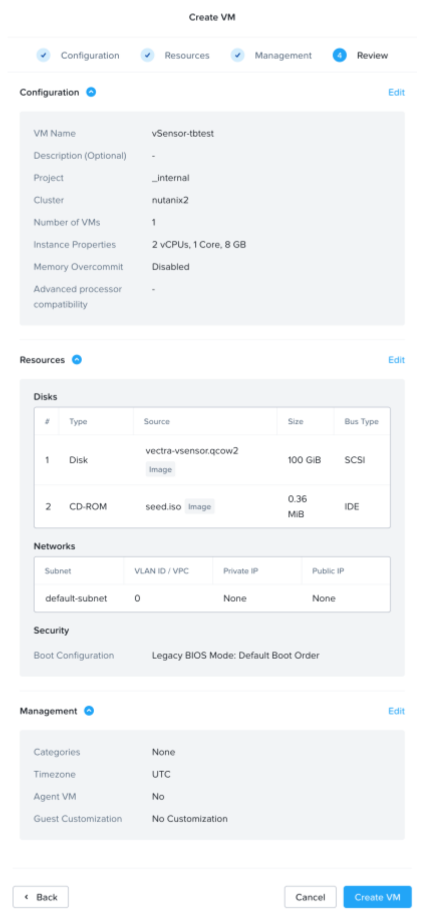
Categories
Type to search...
Tag the VM with Category: Value to assign policies associated with value

Timezone
(UTC) UTC
Use UTC timezone for Linux VMs and local timezone for Windows VMs.

Use this VM as an Agent VM

Guest Customization

Script Type	Configuration Method
No Customization	Custom Script



Configuration

VM Name	vSensor-tbtest
Description (Optional)	-
Project	_internal
Cluster	nutanix2
Number of VMs	1
Instance Properties	2 vCPUs, 1 Core, 8 GB
Memory Overcommit	Disabled
Advanced processor compatibility	-

Resources

Disks

#	Type	Source	Size	Bus Type
1	Disk	vectra-vsensor.qcow2 Image	100 GiB	SCSI
2	CD-ROM	seed.iso Image	0.36 MiB	IDE

Networks

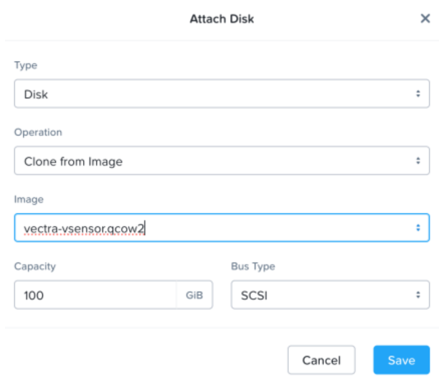
Subnet	VLAN ID / VPC	Private IP	Public IP
default-subnet	0	None	None

Security

Boot Configuration   Legacy BIOS Mode: Default Boot Order

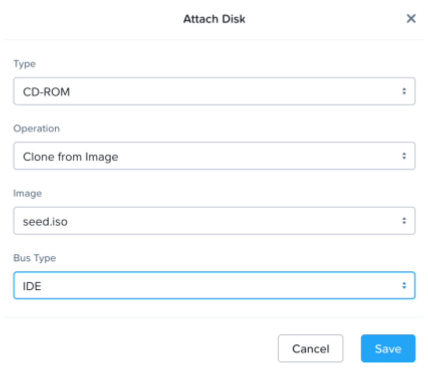
Management

Categories	None
Timezone	UTC
Agent VM	No
Guest Customization	No Customization



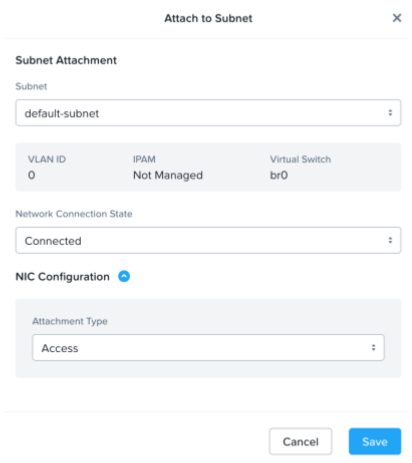
Attach Disk

Type	Disk
Operation	Clone from Image
Image	vectra-vsensor.qcow2
Capacity	100 GiB
Bus Type	SCSI



Attach Disk

Type	CD-ROM
Operation	Clone from Image
Image	seed.iso
Bus Type	IDE



Attach to Subnet

Subnet Attachment

Subnet	default-subnet
VLAN ID	0
IPAM	Not Managed
Virtual Switch	br0
Network Connection State	Connected

NIC Configuration

Attachment Type	Access

Notes on Prism Central vSensor VM Creation:

- Page 1 “Configuration”
  - **Name** – Use a name of your choice.
  - **Description** – Optional.
  - **Project/Cluster** – Set if required for your org.
  - **Number of VMs**
    - Normally 1 if deploying a single vSensor.
    - This can be set to more than 1 if you are deploying multiple identically configured vSensors. The additional vSensors would be named with -1, -2, etc after the configured name.
  - **VM Properties**
    - Refer to Nutanix vSensor Requirements and Throughput for vSensor specifications deploying.
    - **CPUs** – Set to 2, 4, 8, or 16 based on the size vSensor you are deploying.
    - **Cores** – Always set to 1.
    - **Memory** – Set to 8, 8, 16, or 64 based on the size of vSensor you are deploying.
  - **Advanced Settings** – Not required.
- Page 2 “Resources”
  - **Disks** – Add the disks per the screenshots above, cloning them from the uploaded vSensor images.
    - Change the size to match the desirec vSensor specifications.
  - **Network** – Attach to the subnet of your choice.
    - This will be where the Mangement (MGT1) interface of the vSensor will be connected.
    - Initially it will boot with DHCP, but this can be changed to static at the CLI of the vSensor.
      - See Initial vSensor configuration at the Vectra vSensor CLI for details.
  - **Add Mirror Destination NIC** – Click here to add the NIC that will become the vSensor capture port.
  - **Boot Configuration**
    - This must be set to “Legacy BIOS Mode” with “Default Boot Order (CD-ROM, Disk, Network).
    - The seed.iso (CD-ROM) is read during vSensor boot to help configure network adapters.

        - **Shield VM Security Settings** – Not required.
- Page 3 "Management"
    - **Enable 'Default-Storage' Policy**
        - Whether you enable this or not will depend on your defaults.
        - As per the Nutanix vSensor Requirements and Throughput:
            - Vectra recommends that Sensors are configured to use storage local to the hypervisor and are not stored on a SAN. Vectra vSensors require extremely high throughput from their disk storage and this throughput cannot normally be sustained by SAN systems without impact to other SAN users.
        - Also keep in mind that you will be pinning (setting affinity) for each vSensor to a specific node in each cluster.
    - **Categories**
        - This should be left blank for Traffic Mirroring deployments.
    - **Timezone** – Configure per your standards.
    - **Use this VM as an Agent VM**
        - The vSensor typically should be set as an Agent VM to ensure that it boots before other VMs on the node and traffic capture is as complete as possible..
        - Enable this checkmark to set the VM as an agent VM.
    - **Guest Customization** – Not required.
- Page 4 "Review"
    - Review the settings, go back and change anything if required, and when ready, click "Create VM".

2. Set VM Affinity

Affinity ties the vSensor VM to a specific node so that is not moved from a node and setting it is a best practice

- This is setting an affinity to a specific node (also known as pinning).
- Since traffic mirroring sessions cannot mirror host ports across nodes, pinning the VM prevents the vSensor from moving to a location where the mirroring session would no longer function.

To set affinity using the ACLI.

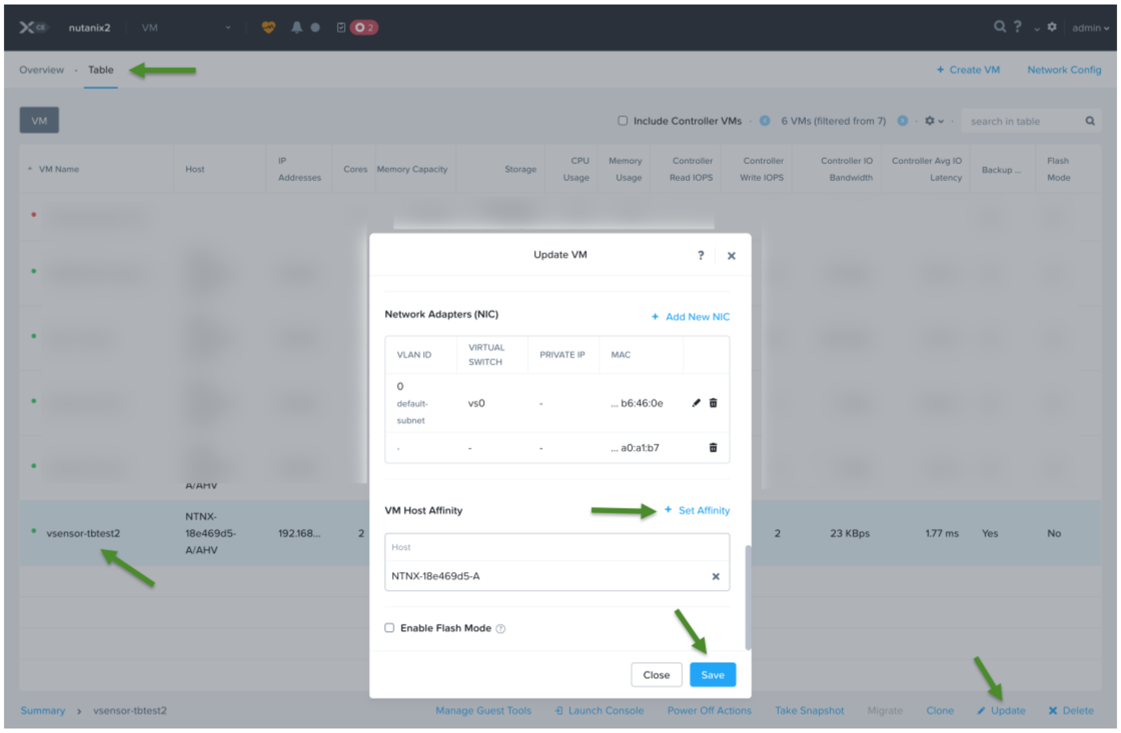
- Log in to a CVM in your target cluster and access the ACLI.
- Execute the below ACLI command. vSensor_Name should be replaced by your vSensor name and x.x.x.x should be the IP of the node you are setting affinity to.

```
nutanix@CVM $ acli
<acropolis> vm.affinity_set vSensor_Name host_list=X.X.X.X
```

Affinity can also be set in the GUI without needing to find the IP of the node you are setting affinity to:

- Navigate in your Prism Element UI (for the cluster you are working on) to the VM table view, select your vSensor VM and click the "Update" button.
- Scroll down to the bottom and set the affinity to the specific node you desire.

© 2025 Vectra AI, Inc.



vsensor-tbtest2 | NTNX-18e469d5-A/AHV | 192.168... | 2 | 2 | 23 KBps | 1.77 ms | Yes | No

Network Adapters (NIC)

VLAN ID	VIRTUAL SWITCH	PRIVATE IP	MAC
0 default-subnet	vs0	-	... b6:46:0e
-	-	-	... a0:a1:b7

VM Host Affinity

Host: NTNX-18e469d5-A

☐ Enable Flash Mode

3. Perform Shared Steps After Deployment

The vSensor is now deployed in Nutanix and you can now complete the Shared Steps After vSensor Deployment.

4. Create a Traffic Mirroring Session

Please see Traffic Mirroring for full guidance from Nutanix with additional detail. This guide will contain only high-level steps for creating a session.

To create a Traffic Mirroring Session:

- In Prism Central, navigate to *Infrastructure* > *Network & Security* > *Network Services* > *Traffic Mirroring.*
- Click “Create Mirror Session”.
- On the 1st screen, give your session a name, optional description, choose the cluster, and virtual switch to use for the mirrored traffic.
  - **Note:** Nutanix recommends utilizing a non-default virtual switch with a Maximum Transmission Unit (MTU) configured between 1600 and 9000 for traffic mirroring to a destination on the remote host. You can also use the default virtual switch **VS0** for traffic mirroring with an MTU set to 1600 bytes. For more information on MTU, see Requirements and Limitations of Flow Virtual Networking and Configuring Virtual Switch for VPC Traffic Types.
- On the 2nd screen, you can the source types, host ports, and VM interfaces that you wish to mirror on the Source side, and on the Destination side you will choose the vSensor you are using as the destination and its Mirror Destination NIC.
  - Only VMs with Mirror Destination NICs that are not already mapped will be shown as options.
  - For Direction, typically you will want to choose “Both” to capture both ingress and egress traffic.
- One the 3rd summary screen, validate your choices and “Create and Enable Session”.
- Repeat these steps for other vSensors as required. Sessions can be updated by editing them.

Shared Steps After vSensor Deployment

These steps are shared fully for both Service Chaining 1.0 and Traffic Mirroring deployment methods:

1. Power on your vSensor.
    a. The Sensor will do an initial boot and then shortly reboot and use the seed.iso file to perform some initial configuration. The seed.iso contains init-config which is required for the vSensor to detect the vNIC and assign a DHCP or static IP to the management (MGT1) interface.
    b. If you miss the reboot, you can do a reboot at the CLI of the vSensor if you are unsure that it already happened.
2. Perform initial configuration at the Sensor's command line via SSH or hypervisor console to set a static IP, DNS, and change the vSensor default password.
3. Pair the vSensor with your Vectra Brain.
4. Validate that traffic is flowing to the vSensor capture interface.

1. Power on Your vSensor VM

You can now power on your vSensor VM. If you have your hypervisor console open, you can watch the boot process. You will see an ubuntu boot screen from which you can hit esc to watch system messages during the boot process if you desire.

- The Sensor will do an initial boot and then shortly reboot and use the seed.iso file to perform some initial configuration.
    - The seed.iso contains init-config which is required for the vSensor to detect the vNIC for MGT and assign DHCP or static IP.
- If you miss the reboot, you can do a reboot at the CLI of the vSensor if you are unsure that it already happened. The reboot command is "**`reboot`**".

Once you've completed the initial boot and reboot, you are ready to login and do the initial vSensor configuration at the CLI.

- The initial username is: vectra
- The initial password is: changethispassword

2. Initial vSensor Configuration at the Vectra vSensor CLI

If you are not using DHCP or would like to set a static address, you will need to login to the CLI of the vSensor to set a static interface assignment. DNS for vSensors must also be configured at the CLI.

vSensor login at the CLI is very similar to logging in to physical Sensors. The primary difference is that there is no physical serial console, IPMI/iDRAC, or other ports to log in to. Logging in can be done via your hypervisor console function or using SSH to the management port if it was preconfigured with DHCP.

- Connect to your vSensor CLI using your hypervisor console or "**`ssh vectra@<IP or Hostname>`**" if you use DHCP and already know the address or hostname.
    - If the vSensor has shown up in the Vectra UI then you should be able to see its IP address in *Data Sources > Network > Sensors* as well.

- !! Please note: Occasionally you may need to switch to a different console to get the OS Kernel console shown that will allow login. When this happens, you can use *ALT + Arrow keys* to change the console to one that allows you entry to the CLI. Nutanix deployments may not pass the *Alt + Arrow keys* to change console. Switching to a different browser such as Firefox has worked correctly for many customers. Please see Device Console doesn't progress past “Loading Initial RamDisk…” during boot-up for more details.

- Once logged in to the appliance you can view command syntax for the "set interface" command:

```
set interface -h
Usage: set interface [OPTIONS] [mgt1] [dhcp|static] [IP] [SUBNET_MASK]
[GATEWAY_ADDRESS]

Sets mgt1 to either dhcp or static ip configuration

Options:
-h, --help Show this message and exit.
```

- Setting the IP address statically:
  - IPv6 is supported for the MGT1 interface. For full details, including information regarding dual stack support, please IPv6 Management Support for Vectra Appliances on the Vectra support portal. Below we will show how to enable IPv6 support (its off by default) and the syntax to use when setting an IPv4 or IPv6 address.

  - To enable/disable IPv6 support

```
# show ipv6 enabled
IPv6 is disabled

# set ipv6 enabled
Response: ok

# show ipv6 enabled
IPv6 is enabled

# set ipv6 disabled
Response: ok
```

  - Setting IPv4 and IPv6 syntax examples:

    Execute the following command to set the MGT1 interface to the desired static IP address:

```
IPv4 Syntax:
set interface mgt1 static x.x.x.x y.y.y.y z.z.z.z

Where:
x.x.x.x is the desired interface IP address
y.y.y.y is the desired interface network mask
z.z.z.z is the desired gateway

IPv6 Syntax:
set interface mgt1 static [IPv6 IP] [Subnet Mask] [Gateway]

Example:
set interface mgt1 static 2001:0db8:0:f101::25 64 2001:0db8:0:f101::1
```

© 2025 Vectra AI, Inc.

- To change back to DHCP (default):

```
set interface mgt1 dhcp
```

- Configure DNS for the appliance:

  Command syntax to set DNS (up to 3 nameservers are supported):

```
set dns [nameserver1 <ip>] [nameserver2 <ip>] [nameserver3 <ip>]
```

  Example:

```
set dns 10.50.10.101 10.50.10.102
```

  Verifying DNS Configuration:

```
show dns
```

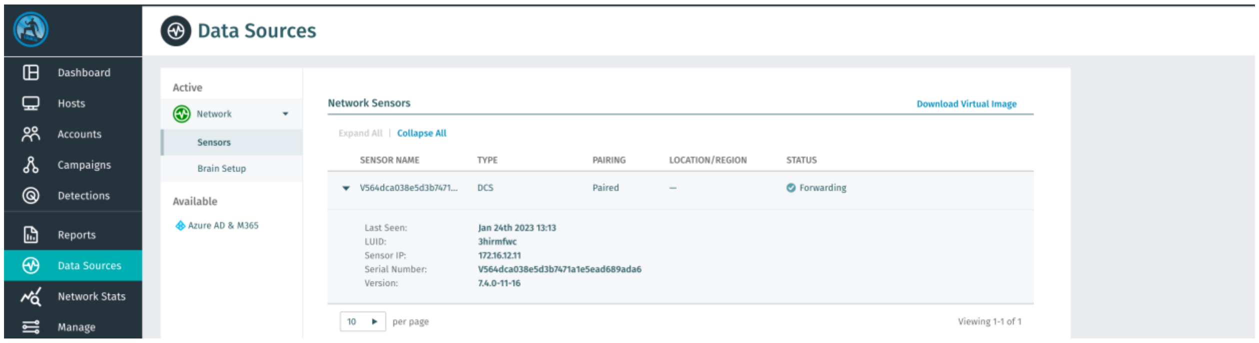
- Once you have set an IP and DNS, please use the "**`set password`**" command to change the password or you may wait and change all paired Sensor passwords en masse in the Vectra UI later at *Data Sources > Network > Sensors > Sensor Configuration > CLI Password (Sensors)* if you wish to keep them in sync.

Example:

```
Welcome to Cognito 6.5.1-1-25, up 24 weeks, 15 hours, 46 minutes (5.4.0-45-generic)

Open source licensing information used in this product is available at https://www.vectra.ai/opensource

Last login: Tue Mar 16 19:06:19 2021 from 172.16.12.1

Welcome to the Vectra Support CLI!

  Model:             DCS
  Mode:              sensor
  Update version:    6.5.1-1-25
  Colossus version: 2:6.5-148-g36b896dc41
  User:              vectra
  Local time:        2021-03-18 18:25:53.223004

  Use 'show commands' to get a list of available commands
  Use 'help' command or '<command> --help' to get help

  Welcome to the Vectra Mobile Attack Lab.  VECTRA CONFIDENTIAL

vscli > set interface mgt1 static 172.16.12.11 255.255.255.0 172.16.12.1
Interfaces updated successfully
vscli > set dns 10.50.10.101
DNS Set: success
vscli > show interface
mgt1:
    Running:
        Gateway: 172.16.12.1,
        Ip: 172.16.12.11,
        Link Speed: 10Gbps,
        Link State: up,
        Mac: 00:0c:29:89:ad:a6,
        Mode: static,
        Netmask: 255.255.255.0
vscli > show dns
Id|Server      |Description
1  10.50.10.101 Configured DNS nameserver
```

3. Pairing Nutanix vSensors

- vSensors do not offer a web UI.
  - The Vectra GUI for vSensor management is located at *Data Sources* > *Network* > *Sensors*.
  - Some configuration of the vSensor can be done at the CLI of the vSensor using the "**`vectra`**" user.
- vSensor images are already associated with the Brain they were downloaded from and will appear in the Brain *Data Sources* > *Network* > *Sensors* page when booted for the first time.
- vSensors are unique to the Brain that the image was downloaded from and cannot normally be paired to other Brains in your environment.
  - If a **"Sensor Registration Token"** is used, a deployed vSensor can be paired to a Brain other than the one the image was originally downloaded from.
  - This is covered in the below Pairing with new or changed Brains section below.
- As per the Vectra Respond UX Deployment Guide or Vectra Quadrant UX Deployment Guide, Sensors, including vSensors, support pairing by IP or by hostname.
  - Pairing by hostname is preferred in failover scenarios. See guide above for more details.
- Once the vSensor is powered on and the interface configuration is set, the vSensor will announce itself to the Brain.
  - This can take a couple of minutes, check firewall rules if there is an issue.
  - If the vSensor appears in the Brain *Data Sources* > *Network* > *Sensors* page, then it has made a successful HTTPS connection to the Brain.
  - If the vSensor does not appear in the Brain *Data Sources* > *Network* > *Sensors* page, check that the vSensor has IP connectivity and that TCP port 443 (HTTPS) is permitted through your firewall.
  - If the vSensor is unable to pair with the Brain, complete its initial update or forward metadata to the Brain check that TCP port 22 (SSH) is permitted through your firewall.
- If the announce is successful, the vSensor will appear at the *Data Sources* > *Network* > *Sensors* page.
- If **"Auto Pairing"** is enabled in *Data Sources* > *Network* > *Sensors* > *Sensor Configuration* > *Sensor Pairing and Registration*, the pairing process will begin automatically.
  - Enabling **"Auto Pairing"** is a best practice during rollout.
- If **"Auto Pairing"** is not enabled, the vSensor must be manually paired by clicking on the "Pair" icon .
  - You will then be presented with a dialog box where you can start the pairing process.
- Initially you will see the **"Pairing Status"** as **"Pairing"** once the vSensor has successfully announced itself to the Brain.
  - Once pairing is complete, the **"Pairing Status"** will change to **"Paired"**, and the **"Status"** should change to **"Forwarding"** once traffic is successfully being forwarded from the vSensor to the Brain.



Data Sources

Network Sensors

SENSOR NAME	TYPE	PAIRING	LOCATION/REGION	STATUS
V564dca038e5d3b7471...	DCS	Paired	–	Forwarding

Last Seen: Jan 24th 2023 13:13
LUID: 3hirmfwc
Sensor IP: 172.16.12.11
Serial Number: V564dca038e5d3b7471a1e5ead689ada6
Version: 7.4.0-11-16

10 per page — Viewing 1-1 of 1

- **Please note:**
  - vSensors, like physical Sensors, will update themselves to stay current with their Brain.
  - After pairing, the vSensor will update by receiving an update from the Brain.
    - This process is automatic, and no input is required.
  - Certain vSensor CLI functions and traffic functions will become available only after the vSensor has fully updated.

  - Depending on the specific version of the vSensor, you may see errors or warnings when running CLI functions during the period of time when the vSensor is still updating.
- The vSensor can be renamed or have its location labeled as desired by clicking on the pencil icon on the right of the vSensor.

Pairing With New or Changed Brains

- If you have a backup of your Brain and restore it to a new Brain with the same configuration (IP or hostname), previously paired Sensors (including vSensors) will connect to the new Brain automatically as the Sensor state is saved in the backup.
- If the Brain IP has changed but otherwise remains the same, the vSensors may be updated to the new IP address using the "**`set brain`**" command.
- Existing tunnels have to terminate to re-establish connection to a new Brain. This can be accomplished a few different ways.
  - Naturally, because the original Brain is no longer reachable due to firewall change, hardware or software failure, etc.
  - Unpairing the vSensor from the original Brain and having the vSensor attempting communication to the original Brain.
  - Using the "**`set brain`**" command at the CLI will terminate an existing tunnel and attempt to start pairing with a new Brain.
- If you have a Brain that will not be restored from backup that you wish to pair an existing vSensor to, this is possible via the use of the "Sensor Registration Token".
  - Retrieve or generate a current Sensor Registration Token from *Data Sources > Network > Sensors > Sensor Configuration > Sensor Pairing and Registration* in the Brain GUI.
  - Perform the "**`set registration-token <token>`**" command at the Sensor CLI.
  - Finally perform the "**`set brain <IP or Hostname>`**" command at the Sensor CLI (depending on if you have selected to pair via the management IP or DNS name in *Data Sources > Network > Sensors > Sensor Configuration > Sensor Pairing and Registration).*

vSensors and Pairing by Hostname vs IP

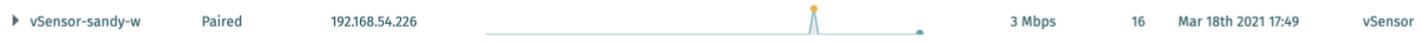
- The vSensor image downloaded from the Brain will use, by default, the Brain's IP address for pairing.
- You will need to set the *Data Sources > Network > Sensors > Sensor Configuration > Sensor Pairing and Registration* **"Pair using DNS name"** option to generate the virtual machine image that points at a hostname.
- When this setting is changed, it does not affect any previously paired (either by IP or Hostname) vSensors.

4. Traffic Validation

Please see the following Vectra support article for recommendations on network traffic that should be examined and excluded from analysis:

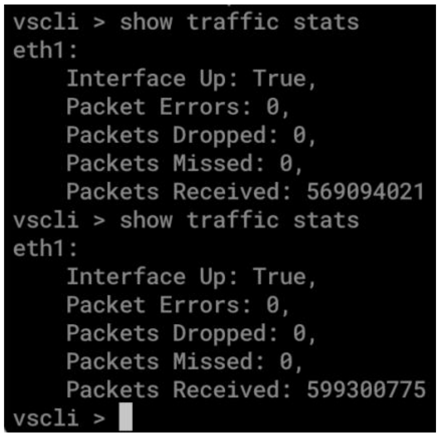
- Vectra Platform Network Traffic Recommendations

For a quick spot check to see that you are receiving any traffic at all via the vSensor you many want to check the GUI and/or CLI for statistics. If the vSensor is seeing more than 1 Mbps of traffic, this will show in the GUI under *Network Stats > Ingested Traffic* after a few minutes.



vSensor-sandy-w   Paired   192.168.54.226   3 Mbps   16   Mar 18th 2021 17:49   vSensor

- You can see traffic flow immediately at the CLI of the Sensor using the "`show traffic stats`" command.
- Execute this command a few times in a row to see increasing packet counts.



```
vscli > show traffic stats
eth1:
    Interface Up: True,
    Packet Errors: 0,
    Packets Dropped: 0,
    Packets Missed: 0,
    Packets Received: 569094021
vscli > show traffic stats
eth1:
    Interface Up: True,
    Packet Errors: 0,
    Packets Dropped: 0,
    Packets Missed: 0,
    Packets Received: 599300775
vscli >
```

After sending traffic to your Sensors, it is a best practice to validate that the traffic observed meets quality standards required for accurate detection and processing. Vectra's Enhanced Network Traffic Validation feature provides alarms and metrics that can be used to validate the quality of your traffic. Please see the following Vectra support article for details:

- Enhanced Network Traffic Validation

Support for Nutanix Management of VMware vSphere (ESXi)

Nutanix environments can use multiple hypervisors such as VMware vSphere ESXi or Hyper-V in addition to its native AHV hypervisor. In such environments, customers will often share overall management of the individual hypervisors and/or VMs running on them between Nutanix Prism and VMware vCenter or the embedded host client for ESXi.

In the case of running ESXi hosts with Nutanix Prism, administrators should use the VMware vSensor image and not the Nutanix vSensor image for deployment. Instructions for deploying the VMware vSensor are available in the VMware vSensor Deployment Guide. The fact that Nutanix is being used for management of the ESXi hosts will be invisible to the VMware vSensor. Vectra's vSensor will not know that it is running in a Nutanix environment. If you are still using vCenter as part of your deployment, the standard vCenter integration described in the VMware vSensor Deployment Guide is still valid.

Nutanix has validated Vectra's VMware vSensor for AOS 6.10 and 7.0.

Appendix

vSensor Deployment Step 3 Alternate API Instructions

This is an alternative API based approach to 3. Create network function provider category and value. It is highly recommended to do this via the Prism Central UI. These instructions were the original required method but since Nutanix now allows this to be done in the PC UI, it is much simpler to just do it there unless you want to fully automate deployment processes programmatically in your environment.

In this section we will create the network function provider category, assign a value to the category, and then verify that assignment. This is done using v3 API calls against your Prism Central server. Some things to note:

Step 3.1 - Create the "network_function_provider" category

```
PUT - https://{{pc_ip}}:9440/api/nutanix/v3/categories/network_function_provider
```

Body:

```
{
    "name": "network_function_provider"
}
```

Response Example (description is an optional value that can be added similarly to "name" as in the above body):

```
{
     "system_defined": false,
     "name": "network_function_provider",
     "description": null
 }
```

Step 3.2 - Assign a value to the category we just created

```
PUT - https://{{pc_ip}}:9440/api/nutanix/v3/categories/network_function_provider/vectra_vsensor
```

Body:

```
{
     "value": "vectra_vsensor"
}
```

Response Example:

```
{
    "system_defined": false,
    "name": "network_function_provider",
    "value": "vectra_vsensor",
    "description": "for monitoring vendor"
}
```

In the above example, "vectra_vsensor" was chosen but this can be any name you desire. It is suggested that the name be the vendor's name or application name of the network function VM.

Step 3.3 - Verify the category and value for the network_function_provider

```
POST - https://{{pc_ip}}:9440/api/nutanix/v3/categories/network_function_provider/list
```

Body:

```
{
    "kind": "category"
}
```

Response Example:

```
{
    "api_version": "3.1",
    "metadata": {
        "total_matches": 2,
        "kind": "category",
        "length": 2,
        "offset": 0
    },
    "entities": [
        {
            "description": "",
            "value": "firewall",
            "system_defined": false
        },
        {
            "description": "",
            "value": "vectra_vsensor",
            "system_defined": false
        }
    ]
}
```

In this response example there are two network_function_provider categories (firewall and vectra_vsensor).

vSensor Deployment Step 4 Alternate API Instructions

In order to create a network function chain in the next step, we first need to retrieve the name and UUID of the AHV cluster where you will deploy the vSensor and network function chain. This is NOT your Prism Central server.

```
POST - https://{{pc_ip}}:9440/api/nutanix/v3/clusters/list
```

Body:

```
{
    "kind": "cluster"
}
```

Abbreviated response example:

```
.
.
.
            "spec": {
                "name": "nutanix2",
                "resources": {
                    "config": {
                        "software_map": {
.
.
.
"metadata": {
                "last_update_time": "2022-06-15T17:59:38Z",
                "kind": "cluster",
                "uuid": "0005dd45-7584-73d2-07af-52540083d046",
                "spec_version": 0,
                "creation_time": "2022-06-15T17:59:38Z",
                "categories_mapping": {},
                "categories": {}
.
.
.
```

vSensor Deployment Step 6 Alternate API Instructions

This is an alternative API based approach to the later parts of 6. Update the Powered off VM Using Both the ACLI and Prism Central UI. It is highly recommended to do this via the Prism Central UI. These instructions were the original required method but since Nutanix now allows this to be done in the PC UI, it is much simpler to just do it there unless you want to fully automate deployment processes programmatically in your environment.

Step 6.2 - Get the UUID of the network function VM (vSensor)

```
<acropolis> vm.list
VM name                    VM UUID
PC01                       001ea51a-a84c-4484-ac6c-aa6dd5a4ce17
debian_vm                  97ffead9-58e0-4c09-9dac-ec3faafbf5b7
debian_vm_demo             7b2591ec-b673-43a7-94c0-fc11f47d26c5
vSensor-Demo               9e15edae-3b57-4886-9b8d-3e77b79dcb45
vectra vsensor api test    8896756f-4eba-43f5-80ab-d3353acd7bc9
vectra_vsensor_chain_test  90893122-046b-447d-b02d-246f9843ddb3
```

Step 6.3 - GET all values for all parameters of the network function VM (vSensor)

Any REST API call that updates an existing entity requires that you perform a GET on the value before updating it to retrieve the most current value of the spec. Failure to get the most recent spec before an update will lead to failed tasks in Prism Central.

In our demo example, the vSensor we are using is vSensor-Demo so we will make the API call for that VM UUID with an empty body. The response example is an abbreviated response and only includes the "spec" portion of the response. This "spec" portion of the response will need to be modified to be used as the body of the PUT API call in the next step. See step 4 below for more details.

```
GET - https://{{pc_ip}}:9440/api/nutanix/v3/vms/9e15edae-3b57-4886-9b8d-3e77b79dcb45
```

Abbreviated response example (already modified to be the body of the PUT in the next step):

```
 {
    "spec": {
        "name": "vSensor-Demo",
        "resources": {
            "num_threads_per_core": 1,
            "vnuma_config": {
                "num_vnuma_nodes": 0
            },
            "serial_port_list": [],
            "nic_list": [
                {
                    "nic_type": "NORMAL_NIC",
                    "uuid": "58e9b04c-b5d6-4aa7-8619-14247af5c285",
                    "ip_endpoint_list": [],
                    "vlan_mode": "ACCESS",
                    "mac_address": "50:6b:8d:dd:a9:7d",
                    "subnet_reference": {
                        "kind": "subnet",
                        "name": "default",
                        "uuid": "56deddea-3df9-47a0-ab9a-45348673b261"
                    },
                    "is_connected": true,
                    "trunked_vlan_list": []
                },
                {
                    "nic_type": "NETWORK_FUNCTION_NIC",
                    "uuid": "2bdd5afe-b39b-4a08-8547-036ecf32ce7e",
                    "network_function_nic_type": "TAP",
                    "vlan_mode": "ACCESS",
                    "mac_address": "50:6b:8d:e0:bd:bc",
                    "is_connected": true,
                    "trunked_vlan_list": []
                }
            ],
            "num_vcpus_per_socket": 1,
            "num_sockets": 2,
            "disable_branding": false,
            "enable_cpu_passthrough": false,
            "gpu_list": [],
            "is_agent_vm": true,
            "memory_size_mib": 8192,
            "boot_config": {
                "boot_device_order_list": [
                    "CDROM",
                    "DISK",
                    "NETWORK"
                ],
                "boot_type": "LEGACY"
            },
            "hardware_clock_timezone": "UTC",
            "power_state_mechanism": {},
            "power_state": "OFF",
            "machine_type": "PC",
            "vga_console_enabled": true,
            "disk_list": [
                {
                    "uuid": "17f43679-913c-4d8c-8f91-55158b7df1ec",
                    "disk_size_bytes": 107374182400,
                    "storage_config": {
                        "storage_container_reference": {
                            "kind": "storage_container",
                            "uuid": "4adfaee8-5d13-43d0-9e2c-0acb78c20bf3",
                            "name": "default-container-36061588278502"
                        }
                    },
                    "device_properties": {
                        "disk_address": {
                            "device_index": 0,
                            "adapter_type": "SCSI"
                        },
                        "device_type": "DISK"
                    },
                    "data_source_reference": {
                        "kind": "image",
                        "uuid": "49c30577-d09f-4df0-8308-9f631127cad8"
                    },
                    "disk_size_mib": 102400
                },
                {
                    "uuid": "c2817978-d7a7-4bd7-80ef-eb6c14daa7ac",
                    "disk_size_bytes": 377856,
                    "storage_config": {
                        "storage_container_reference": {
                            "kind": "storage_container",
                            "uuid": "4adfaee8-5d13-43d0-9e2c-0acb78c20bf3",
                            "name": "default-container-36061588278502"
                        }
                    },
                    "device_properties": {
                        "disk_address": {
                            "device_index": 0,
                            "adapter_type": "IDE"
                        },
                        "device_type": "CDROM"
                    },
                    "data_source_reference": {
                        "kind": "image",
                        "uuid": "7845607e-2f43-4ba1-a87c-f0a8d2b9e7cb"
                    },
                    "disk_size_mib": 1
                }
            ]
        },
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "0005dd45-7584-73d2-07af-52540083d046"
        }
    },
    "api_version": "3.1",
    "metadata": {
        "last_update_time": "2022-08-22T17:14:43Z",
        "kind": "vm",
        "uuid": "9e15edae-3b57-4886-9b8d-3e77b79dcb45",
        "project_reference": {
            "kind": "project",
            "name": "default",
            "uuid": "6ec5e4cf-81da-4c2e-8ee2-efa5bbf522f1"
        },
        "creation_time": "2022-08-16T19:41:12Z",
        "spec_version": 5,
        "categories_mapping": {
            "network_function_provider": [
                "vectra_vsensor"
            ]
        },
        "entity_version": "5",
        "owner_reference": {
            "kind": "user",
            "uuid": "00000000-0000-0000-0000-000000000000",
            "name": "admin"
        },
        "categories": {
            "network_function_provider": "vectra_vsensor"
        }
    }
}
```

!!Please note: The initial get is not expected to return a value under “categories”.

© 2025 Vectra AI, Inc.

Step 6.4 - Add the network function provider category and value to the vSensor VM specifications.

The entire, and only the "spec" portion of the response from step 8.3, needs to be modified to add the network function provider category and value to the VM specification in the metadata section near the bottom of the response.

*** **Note:** Modify and use only the spec section of the VM data. Do not copy any of the other response sections. If the format is not set correctly then the API will fail to execute, and you may get a 422 “Additional properties are not allowed…” error message. ***

We modified the "metadata" portion of the response above by adding the following category and value that we created earlier:

```
         ,
        "categories": {
            "network_function_provider": "vectra_vsensor"
        }
```

Then we use the PUT below with the PC API to add this to the VM specification of the network function VM (vSensor).

```
PUT - https://{{pc_ip}}:9440/api/nutanix/v3/vms/9e15edae-3b57-4886-9b8d-3e77b79dcb45
```

Body:

Use the modified "spec" portion of the response from step 8.3 that has been modified as instructed.

Abbreviated response example (we are only showing that it is “PENDING” and you can check the task status):

```
{
    "status": {
        "state": "PENDING",
        "execution_context": {
            "task_uuid": "fbca45b1-f8ab-451a-9758-b62e1026435e"
        }
    },
```

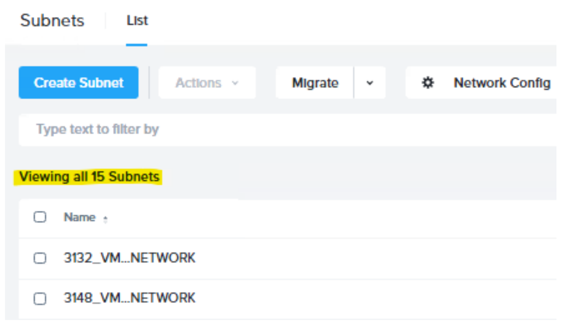
vSensor Deployment Step 7 Alternate API Instructions

This is an alternative API based approach to 7. Direct Traffic to the Configured Network Function Chain. These instructions were the original required method but remain if you want to fully automate deployment processes programmatically in your environment.

Step 7.1 - List Available Networks (Subnets)

This step must be done via Prism Central v3 API calls. In this step we will obtain the UUID of a subnet that should be directed through the network function chain.

- Please note that the Prism Central v3 api defaults to returning a maximum of 20 results unless the body or the request is modified to include “**length**” and “**offset**” parameters which can also be used to paginate results. Our example below does not include these parameters and just uses a default query.
    - Up to 500 subnet entities (0-499 for example) can be retrieved by a single query by using a length of 500 and an offset of 0.
    - “**total_matches**” will also return the total number of subnets available. The Prism Central UI can also show the total number of subnets:



3132_VM...NETWORK
3148_VM...NETWORK

- Nutanix provides more detail about how these parameters work, along with examples in this KB article.

```
POST - https://{{pc_ip}}:9440/api/nutanix/v3/subnets/list
```

Body:

```
{
    "kind": "subnet"
}
```

Response Example:

- Our example contains only one “default-subnet” entity, but customer deployments will typically have many subnets. Find the UUIDs for all subnets that you wish to have mapped to the network function chain.
    - Each node in the cluster will forward traffic they see on these subnets to the vSensor VMs.

```
{
    "api_version": "3.1",
    "metadata": {
        "kind": "subnet",
        "offset": 0,
        "length": 1,
        "total_matches": 1
    },
    "entities": [
        {
            "status": {
                "name": "default-subnet",
                "cluster_reference": {
                    "kind": "cluster",
                    "name": "nutanix2",
                    "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
                },
                "resources": {
                    "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755",
                    "vswitch_name": "br0",
                    "network_function_chain_reference": {
                        "kind": "network_function_chain",
                        "uuid": "9385b9d5-d2cf-4039-b3b1-45f5ef744e1a"
                    },
                    "vlan_id": 0,
                    "subnet_type": "VLAN",
                    "ip_usage_stats": {
                        "num_macs": 5
                    }
                },
                "state": "COMPLETE",
                "execution_context": {
                    "task_uuid": [
                        "8171576a-6bc5-4bc1-b191-2ba6a8cb2f42"
                    ]
                }
            },
            "metadata": {
                "uuid": "f5d8c6bb-8ad9-42c5-b977-8bafbba10b81",
                "spec_hash": "00000000000000000000000000000000000000000000000000",
                "spec_version": 2,
                "owner_reference": {
                    "kind": "user",
                    "uuid": "71f50668-79f9-50d4-8dec-eaaf5a411f36",
                    "name": "tbilen"
                },
                "categories": {},
                "categories_mapping": {},
                "creation_time": "2025-09-10T20:39:29Z",
                "last_update_time": "2025-09-15T21:25:45Z",
                "kind": "subnet"
            },
            "spec": {
                "resources": {
                    "subnet_type": "VLAN",
                    "vswitch_name": "br0",
                    "network_function_chain_reference": {
                        "kind": "network_function_chain",
                        "uuid": "9385b9d5-d2cf-4039-b3b1-45f5ef744e1a"
                    },
                    "vlan_id": 0,
                    "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755"
                },
                "cluster_reference": {
                    "kind": "cluster",
                    "name": "nutanix2",
                    "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
                },
                "name": "default-subnet"
            }
        }
    ]
}
```

Step 7.2 - Get the Subnet Details Using the Retrieved UUID

- Use an empty body in your request.

```
https://{{pc_ip}}:9440/api/nutanix/v3/subnets/f5d8c6bb-8ad9-42c5-b977-8bafbba10b81
```

Response Example:

```
{
    "api_version": "3.1",
    "metadata": {
        "last_update_time": "2025-09-15T21:25:45Z",
        "kind": "subnet",
        "uuid": "f5d8c6bb-8ad9-42c5-b977-8bafbba10b81",
        "spec_version": 2,
        "creation_time": "2025-09-10T20:39:29Z",
        "categories_mapping": {},
        "categories": {},
        "owner_reference": {
            "kind": "user",
            "uuid": "71f50668-79f9-50d4-8dec-eaaf5a411f36",
            "name": "tbilen"
        },
        "spec_hash": "00000000000000000000000000000000000000000000000000"
    },
    "spec": {
        "resources": {
            "subnet_type": "VLAN",
            "vswitch_name": "br0",
            "vlan_id": 0,
            "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755"
        },
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
        },
        "name": "default-subnet"
    },
    "status": {
        "name": "default-subnet",
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
        },
        "resources": {
            "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755",
            "vswitch_name": "br0",
            "network_function_chain_reference": {
                "kind": "network_function_chain",
                "uuid": "9385b9d5-d2cf-4039-b3b1-45f5ef744e1a"
            },
            "vlan_id": 0,
            "subnet_type": "VLAN",
            "ip_usage_stats": {
                "num_macs": 5
            }
        },
        "state": "COMPLETE",
        "execution_context": {
            "task_uuid": [
                "8171576a-6bc5-4bc1-b191-2ba6a8cb2f42"
            ]
        }
    }
}
```

Step 7.3 - Update the Subnet to Reference the Network Function Chain

The "spec" and “metadata” portions (highlighted in yellow above) of the response from step 9.2 need to be copied, modified, and then used as the body of the PUT in this step to update the subnet to point to the network function chain. It is suggested to use an API client that enforces proper json formatting to make the modifications. Some options were given Guidance for using the Nutanix Prism Central REST API. Note that an extra brace needed to be added to the body at the bottom after copying from above.

- The spec portion of the response you copied must be modified to include the NFC reference. This should be done within the resources section of the spec portion of the body.

```
PUT - https://{{pc_ip}}:9440/api/nutanix/v3/subnets/b9aa1db8-47fd-4f39-9b97-d4aeb3d621c3
```

Body:

```
{
    "api_version": "3.1",
    "metadata": {
        "kind": "subnet",
        "uuid": "f5d8c6bb-8ad9-42c5-b977-8bafbba10b81",
        "categories": {},
        "categories_mapping": {},
        "creation_time": "2025-09-10T20:39:29Z",
        "last_update_time": "2025-09-10T20:39:29Z",
        "spec_hash": "00000000000000000000000000000000000000000000000000",
        "spec_version": 0
    },
    "spec": {
        "resources": {
            "subnet_type": "VLAN",
            "vswitch_name": "br0",
            "network_function_chain_reference": {
                "kind": "network_function_chain",
                "name": "vectra_tap",
                "uuid": "9385b9d5-d2cf-4039-b3b1-45f5ef744e1a"
            },
            "vlan_id": 0,
            "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755"
        },
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
        },
        "name": "default-subnet"
    }
}
```

Response example:

```
{
    "metadata": {
        "last_update_time": "2025-09-10T20:39:29Z",
        "use_categories_mapping": false,
        "kind": "subnet",
        "uuid": "f5d8c6bb-8ad9-42c5-b977-8bafbba10b81",
        "spec_version": 2,
        "creation_time": "2025-09-10T20:39:29Z",
        "spec_hash": "00000000000000000000000000000000000000000000000000",
        "categories_mapping": {},
        "categories": {}
    },
    "api_version": "3.1",
    "spec": {
        "resources": {
            "subnet_type": "VLAN",
            "vswitch_name": "br0",
            "network_function_chain_reference": {
                "kind": "network_function_chain",
                "name": "vectra_tap",
                "uuid": "9385b9d5-d2cf-4039-b3b1-45f5ef744e1a"
            },
            "vlan_id": 0,
            "virtual_switch_uuid": "7925c9f7-c20f-44a8-b4e3-e4067048a755"
        },
        "cluster_reference": {
            "kind": "cluster",
            "name": "nutanix2",
            "uuid": "000639fc-7386-2da4-7938-bc2411a17fee"
        },
        "name": "default-subnet"
    },
    "status": {
        "state": "PENDING",
        "execution_context": {
            "task_uuid": "9dd87309-bcd8-4497-9237-029b57f4f28e"
        }
    }
}
```

Worldwide Support Contact Information

- Support portal: https://support.vectra.ai/
- Email: support@vectra.ai (preferred contact method)
- Additional information: https://www.vectra.ai/support

© 2025 Vectra AI, Inc.