Pluribus Networks

Netvisor® ONE

REST API Guide Version 5.1.2

February 2020

Table of Contents

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR PLURIBUS NETWORKS REPRESENTATIVE FOR A COPY.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE ARE PROVIDED “AS IS” WITH ALL FAULTS. PLURIBUS NETWORKS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL PLURIBUS NETWORKS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA, ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF PLURIBUS NETWORKS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses used in this document are not intended to be actual addresses. Any examples, command display output, and figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses in illustrative content is unintentional and coincidental.

Netvisor® OS APIs

Informational Note: The target audience for this guide is experienced API programmers.

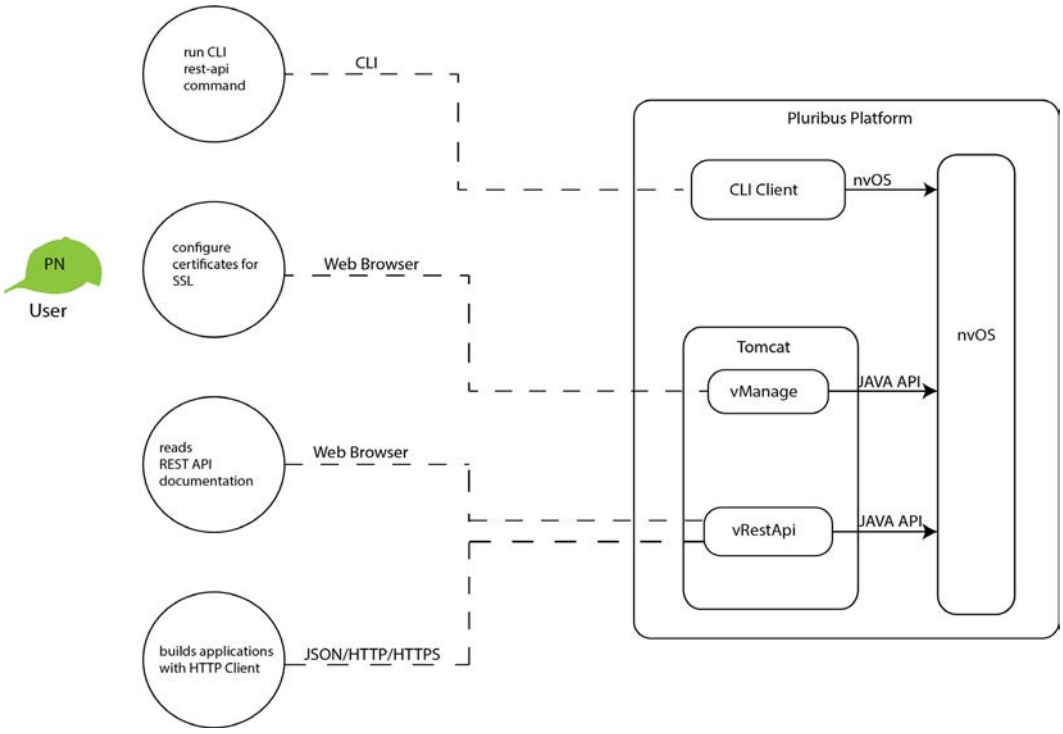
Overview

Representational State Transfer (REST) is a well known method of building Web services over HTTP or HTTPS. Netvisor ONE currently supports this method of building Web services using the data format JavaScript Object Notation (JSON).

You should understand Universal Resource Indicators (URIs) and the JSON schemas for REST APIs before attempting to use them with Netvisor ONE. To build applications, you can use a HTTP client with REST API URIs and JSON schema.

Figure 1 User Interaction with REST API describes how you interact with Netvisor ONE to build applications with the REST API and the components used during the process. The new component, vRestApi, is a war file deployed in Tomcat. vRestApi implements the REST API Web service and interacts with Netvisor ONE using the existing Java API.

Figure 1:User Interaction with REST API



REST API Design

The REST API defines a contract based on HTTP verbs, URIs, and a JSON schema. REST API Clients use HTTP or HTTPS with HTTP verbs and URIs listed in the contract and handle the request and response payloads according to the JSON format for the URIs.

For example, if you want to configure a REST API operation to lookup a vFlow, it should have the following format:

```
GET /vflows/name/DHCP-client

{{"data":{"name":"DHCP-client","scope":"local","type":"system",
     "ether-type":"ipv4","dst-port":67,"proto::"udp","precedence":5,"action":
     "copy-to-cpu"}, {"result":{"SUCCESS"}}}
```

The REST API client reads the documentation to understand what HTTP verb and URI is used to look up a vFlow and also the format of the JSON request and response payloads. In the example, `GET` is the HTTP verb, and the `URI` is `vflows/name/my-vflow` which identifies a resource: a vFlow uniquely identified as `my-flow`. The request does not have a payload while the response contains the my-vflow data and the result status is in JSON format.

The key elements of a RESTful implementation are as follows:

- **Resources** – The first key element is the resource itself. You want to enable an SNMP trap on a switch, and the URL of the switch is http://<*switch-ip*>. In order to enable an SNMP trap using REST, you can issue the command http://<*switch-ip*>/vREST/snmp-trap-enable. This command enables the trap in the Parameter field.
- **Request Verb**s - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example `http://<switch-ip>/vRest/snmp-trap-enable`, the Web browser is issuing a POST Verb because you want to enable a specific trap.
- **Request Headers** – These are additional instructions sent with the request. These might define the type of response required or the authorization details.
- **Request Body** - Data is sent with the request. Data is normally sent in the request when a POST request is sent to the switch. In a POST call, the client actually tells the switch that it wants to add a resource to the switch. Therefore, the request body has the details of the required resource added to the switch.
- **Response Body** – This is the main body of the response and returns the details of the request.
- **Response Status codes** – These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

Restful Methods

You have a RESTful Web service defined at http://<*switch-i*p>/snmp-trap-enable. When the client requestsa Web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. The following is a list of actions, If the respective verbs are sent by the client.

- **POST** — used to create a configuration using the RESTful Web service
- **GET** — used to get a list of all configured parameters using the RESTful Web service
- **PUT** — used to update configuration using the RESTful Web service
- **DELETE** —used to delete a configuration using the RESTful web service

HTTP Response Codes

The REST API follows the conventions of the RESTful style for HTTP response status codes. The following is a list of status codes for known use cases:

- **200** (OK) — successful completion of the REST API operation.
- **201** (Created) — successful completion of a new resource completion.
- **202** (Accepted) — successful start of an asynchronous action.
- **401** (Unauthorized) — unsuccessful authentication with Netvisor ONE.
- **403** (Forbidden) — authentication is successful but you are not authorized to access a resource as defined by Netvisor ONE authorization permissions.
- **404** (Not Found) — the URI cannot map to a resource. You may see this message returned by Apache CXF runtime.
- **415** (Unsupported Media Type) — media type is unsupported. You may see this returned by Apache CXF runtime if a request for application/xml, for example, is received.
- **500** (Internal Server Error) — indicates a general server-side problem with either PN REST API servlet or Netvisor ONE.

Response Payloads

All Netvisor ONE REST APIs return a response payload with the following schema:

```
{{"data":{},result:{"responseCode":integer,"status":"SUCCESS"|"FAILURE","cod
    e":integer,"message":string}}}
```

The response payload may contain `data`, and always contains `result`. `data` consists of one or more resources of the same type. `result` always contains `responseCode` which is a valid HTTP response status code and `status` which is the status of the back-end operation on Netvisor ONE as either `SUCCESS` or `FAILURE`. `result` may contain a code or a message that provides further information about the back-end operation on Netvisor ONE such as next values or an error.

REST API clients should check both the HTTP status code returned in the response Status-Line and the `responseCode` returned in `result`. If the status code in the response Status-Line is not 200, the client does not check `result`. Otherwise `result` should be checked. This is necessary because the HTTP response is streamed back to the REST API client, and an error may occur after a response code of 200 in the response Status-Line is already written to the REST API client. In this case, `data` may contain a partial result. Another example is for a successful `create` operation where the `responseCode`, 201, is returned as the result.

REST API Clients

The REST API does not have a client-side library. When clients use HTTP or HTTPS with the URIs listed in the contract and then handle request and response payloads that follow the JSON schema as described with each URI in the contract. Usually, REST APIs are implemented within Web frameworks that facilitate the use of URIs and the handling of JSON. It is not required for the REST API client to run on the Pluribus Networks switch.

Documentation is provided using Swagger. You use the Swagger Javascript client in a Web browser connected to the Swagger documentation site hosted by Tomcat on the switch. Swagger is designed specifically for REST API documentation and the Javascript client provides a way to display REST URIs and JSON, and a way to try out a URI. Swagger is also a state-of-the-art application for REST API documentation. This is the most involved approach since you have to run a Swagger Javascript client in your Web browser.

You can also use your favorite HTTP client tool such as cURL.

Setting Up Swagger

The REST API uses Swagger-UI for documentation.

Information about Swagger can be found here: http://swagger.io/

Swagger is a great tool for visualizing and testing REST APIs, but it is relatively new and requires manual steps to get everything working.

1. Download the Swagger UI client zip from here:
https://github.com/swagger-api/swagger-ui/tree/v2.0.24

2. Unzip to a local folder. The Swagger-UI needs to authenticate to the switch using Basic Authentication.
3. To enable Basic Authentication for Swagger-UI, edit the file:
swagger-ui-2.0.24/dist/index.html

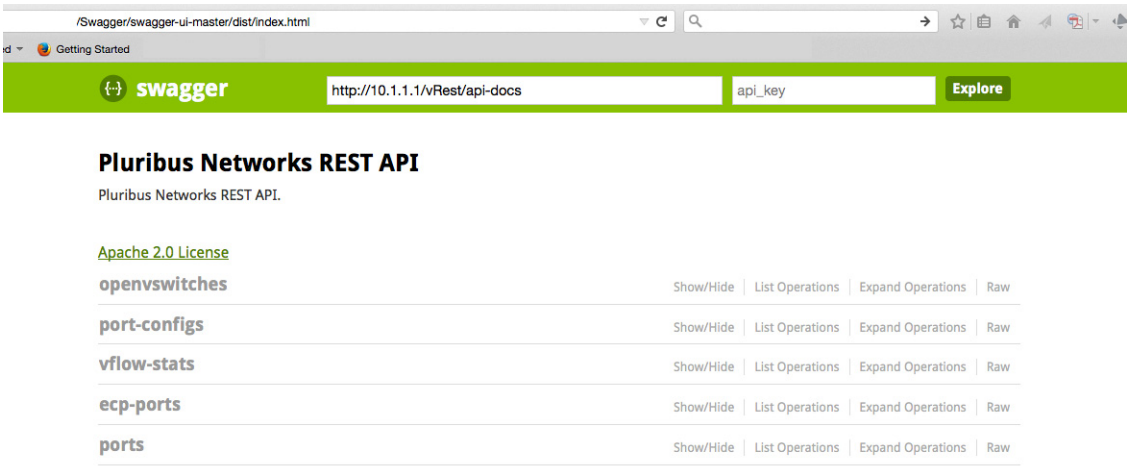
4. Insert the line highlighted in red in the code snippet below:

```
<script type="text/javascript">
$(function () {
  window.authorizations.add("authorization", new
ApiKeyAuthorization("authorization", "Basic bmV0d29yay1hZG1pbjp0ZXN0MTIz",
"header"))
      window.swaggerUi = new SwaggerUi({
      url: "http://petstore.swagger.wordnik.com/api/api-docs",
      dom_id: "swagger-ui-container",
      supportedSubmitMethods: ['get', 'post', 'put', 'delete'],
      onComplete: function(swaggerApi, swaggerUi){
        log("Loaded SwaggerUI");
```

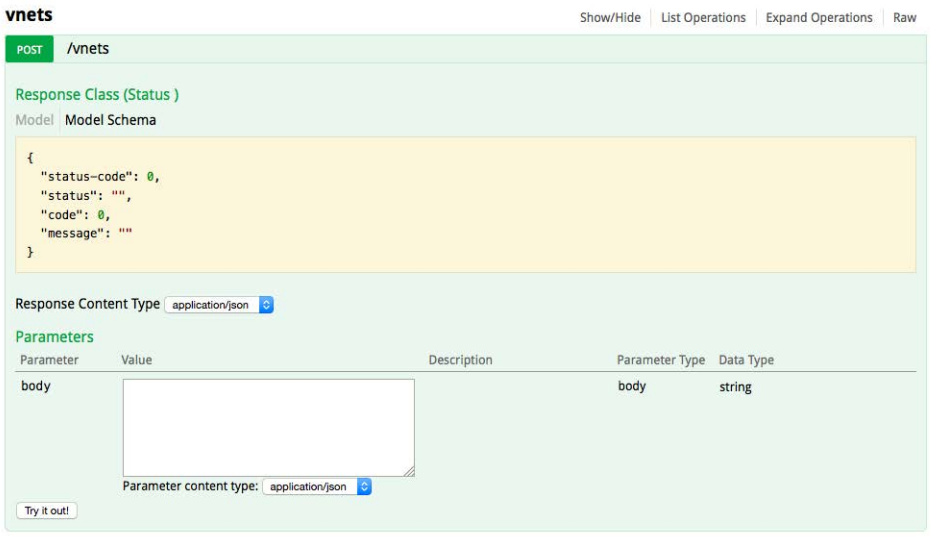
Informational Note: The text in red must be inserted into the code as a single line with no breaks.

This is the base 64 coded credential for network-admin:p1zz@2015. To use another credential, use a base-64 encoding application to encode your `username:password` and substitute it in the above line.

5. To access the Swagger REST API documentation:
   a. Load the file, **swagger-ui-2.0.24/dist/index.html**, into your browser.
   a.
   b. Paste the URL http://<*your-switch-mgmt-ip*>/vRest/api-docs into the Swagger URL field.
   c. Click **Explore.**



You can now browse the various APIs and test them using the Swagger “Try it out!” feature. .



Authentication

The REST API uses Basic Authentication. The username and password sent in Basic Authentication should allow you to log into the CLI. The out-of-box username and password for the switch is network-admin/admin. Here is the corresponding code snippet for swagger-ui:

```
window.authorizations.add("authorization", new
     ApiKeyAuthorization("authorization", "Basic
     bmV0d29yay1hZG1pbjphZG1pbg==", "header"))
```

REST API Examples

In general, numerical values such as port numbers are not enclosed in curly quotes. Otherwise, each parameter and value is enclosed by curly quotes as shown in the examples. You should also have copies of the *Pluribus Networks Netvisor ONE Command Reference A-O* and *Pluribus Networks Netvisor ONE Command Reference P-Z* to identify parameters used by each command.

Example 1: Get a vFlow identified by name:

```
GET /vflows/name/DHCP-client

{{"data":{"name":"DHCP-client","scope":"local","type":"system",
     "ether-type":"ipv4",
     "dst-port":67,"proto":"udp","precedence":"5","action":"copy-to-cpu"},
     {"result":{"SUCCESS"}}}
```

Example 2: Get a vFlow identified by ID:

```
GET /vflows/id/ce46fb

{{"data":{"id":"ce46fb","scope":"local","type":"system","ether-type":"ipv4",
     "dst-port":67,"proto":"udp","precedence":5,"action":
     "copy-to-cpu"}, {"result":{"SUCCESS"}}}
```

Example 3: Create a vflow:

```
POST /vflows {{"data":{"name":"vflow-test","scope":"local","type":"system",
     "ether-type":"ipv4","dst-port":67,"proto":"udp","precedence":"5",
     "action":"copy-to-cpu"}}}

{{"result":{"responseCode":201, "status":"SUCCESS"}}}
```

Example 4: Obtaining user role information:

The REST service, in general, doesn't chase references since the REST client can easily do so using multiple REST API calls.

Here is how to get the role information starting from a user's roles.

1. GET /users/user1/roles

```
{
  "data": [
    {
      "role": "0:1"
    }
  ],
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

2. GET /roles


```
{
  "id": "0:1",
  "name": "read-only-network-admin",
  "scope": "local",
  "vnet-access": "0:0",
  "access": "read-only",
  "running-config": false
},
```


3. Match role from first result set to id from the second result set.

REST API Examples

You can try out these examples in the Swagger UI to ensure that they work on your server-switch. The following Request methods are supported:

GET – Retrieves data from the specified object.

PUT – Adds the supplied information to the specified object; returns a 404 Resource Not Found error if the object does not exist.

POST – Creates the object with the supplied information.

DELETE – Deletes the specified object.

Creating an acl-ips RESTful API

Configure an ACL for denying traffic from the Engineering server to the HR server and name the ACL, `deny-hr`:

POST http://<*switch-ip*>/acl-ips

```
{
  "name": "deny-hr",
  "action": "deny",
  "vlan": "1505",
  "scope": "local",
  "proto": "ip",
  "src-ip": "192.168.10.2",
  "src-ip-mask": 24,
  "src-port": 55,
  "dst-ip": "192.168.200.3",
  "dst-ip-mask": 24,
  "dst-port": 33
}
```

Request URL

http://<*switch-ip*>:80/vRest/acl-ips

Response Body

```
  {
    "result": {
      "status": "Success",
      "code": 0,
      "message": ""
    }
  }
```

Response Code

201

Response Headers

```
  {
    "Content-Type": "application/json"
  }
```

Display the ACL Using REST API

Display the ACL you just configure using the following REST API:

GET http://<*switch-ip*>/acl-ips

```
{"name": "deny-hr"}
```

Request URL

```
http://<switch-ip>:80/vRest/acl-ips
```

Response Body

```
{
  "data": [
    {
      "name": "deny-hr",
      "id": "a000030:27",
      "action": "deny",
      "proto": "ip",
      "src-ip": "192.168.10.2",
      "src-ip-mask": 24,
      "src-port": 55,
      "dst-ip": "192.168.200.3",
      "dst-ip-mask": 24,
      "dst-port": 33,
      "vlan": 1505,
      "scope": "local",
      "port": 0
    }
  ],
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Code

200

Response Headers

```
{
  "Content-Type": "application/json"
}
```

Displaying an ACL Using the Assigned ID

To display an ACL using the assigned ID from the previous example, use the following API:

```
GET http://<switch-ip>/acl-ips/id/a000030:27
```

Request URL

```
http://<switch-ip>:80/vRest/acl-ips/id/a000030%3A27
```

Response Body


```
{
  "data": [
    {
      "name": "deny-hr",
      "id": "a000030:27",
      "action": "deny",
      "proto": "ip",
      "src-ip": "192.168.10.2",
      "src-ip-mask": 24,
      "src-port": 55,
      "dst-ip": "192.168.200.3",
      "dst-ip-mask": 24,
      "dst-port": 33,
      "vlan": 1505,
      "scope": "local",
      "port": 0
    }
  ],
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```


Response Code

200

Response Headers


```
{
  "Content-Type": "application/json"
}
```


Displaying an ACL Using the ACL Name

To display an ACL by name, use the following API:

```
GET http://<switch-ip>/vRest =/acl-ips/name/deny-hr
```

Request URL

```
http://<switch-ip>:80/vRest/acl-ips/name/deny-hr
```


```
{
  "data": [
    {
      "name": "deny-hr",
      "id": "a000030:28",
      "action": "deny",
      "proto": "ip",
      "src-ip": "192.168.10.2",
      "src-ip-mask": 24,
      "src-port": 55,
      "dst-ip": "192.168.200.3",
      "dst-ip-mask": 24,
      "dst-port": 33,
      "vlan": 1505,
      "scope": "local",
      "port": 0
    }
  ],
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```


Response Code

200

Response Headers


```
{
  "Content-Type": "application/json"
}
```


Changing the ACL Action Using the ACL ID

To change the action from deny to permit, use the following API:

```
PUT http://<switch-ip>:80/vRest/acl-ips/id/a000030%3A27
```


```
{"action": "permit"}
```


Request URL

```
http://<switch-ip>:80/vRest/acl-ips/id/a000030%3A27
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Code

200

Response Headers

```
{
  "Content-Type": "application/json"
}
```

Changing the ACL Action Using the ACL Name

To change the action from deny to permit, use the following API:

```
PUT http://<switch-ip>:80/vRest/acl-ips/name/deny-hr

{"action": "permit"}
```

Request URL

```
http://<switch-ip>:80/vRest/acl-ips/name/deny-hr
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Code

200

Response Headers

```
{
  "Content-Type": "application/json"
}
```

Deleting an ACL Using the Assigned ID

To delete an ACL using the assigned ID, use the following API:

```
DELETE http://<switch-ip>/acl-ips/id/a000030:27
```

Request URL

```
http://<switch-ip>/vRest/acl-ips/id/a000030%3A27
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Code

200

Response Headers

```
{
  "Content-Type": "application/json"
}
```

Deleting an ACL Using the ACL Name

To delete an ACL using the assigned ID, use the following API:

```
DELETE http://<switch-ip>/acl-ips/id/deny-hr
```

Request URL

```
http://<switch-ip>/vRest/acl-ips/id/deny-hr
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Code

200

Response Headers

```
{
  "Content-Type": "application/json"
}
```

Creating a VNET and Adding Services

In this example, you create a virtual network (VNET) called **internal-net**, and add DHCP services to it. The VNET has a local scope and is on VLAN 111.

To create the VNET, use the following REST API:

```
PUT http://<switch-ip>/vnets

{
  "name": "",
  "scope": "",
  "vlans": "",
  "ports": "",
  "admin": 0,
  "vrg-id": "",
  "num-vlans": "short",
  "managed-ports": "",
  "config-admin": false,
  "vnet-mgr-name": "",
  "vnet-mgr-storage-pool": ""
}
```

Request URL

```
http://<switch-ip>:80/vRest/vnets

{
  "name": "internal-net",
  "scope": "local",
  "vlans": "111"
}
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": "Vnet created"
  }
}
```

Response Code

201

Response Header

```
{
  "Content-Type": "application/json"
}
```

CLI Show Output

```
CLI server-switch > vnet-show format all layout vertical

  switch:                 pubdev02
  id:                     a000024:1
  name:                   internal-net
  scope:                  local
  vrg:                    internal-net-vrg
  num-vlans:              1
  vlans:                  111
  managed-ports:          none
  admin:                  internal-net-admin
  vnet-mgr-name:          internal-net-mgr
  global:                 false
```

Configuring IP Address Pools

You can configure IP address pools to use with DHCP implementation. In this example, you can see how to configure an IP address pool named **internal-ip-pool**, with the IP address range of **172.16.21.0/24** on VNET ID, `a000024:1`.

Configuring the Initial IP Address Pool

The IP Pool API requires the VNET ID and not the VNET name. The VNET ID is displayed in the show output:

```
CLI server-switch > vnet-show format all layout vertical

  switch:                 pubdev02
  id:                     a000024:1
  name:                   internal-net
  scope:                  local
  vrg:                    internal-net-vrg
  num-vlans:              1
  vlans:                  111
  managed-ports:          none
  admin:                  internal-net-admin
  vnet-mgr-name:          internal-net-mgr
  global:                 false
```

You can create the IP pool using the `network` parameter or create a range of IP addresses, but you cannot use both parameters.

To configure the IP address pool using REST API, use the following API:

```
POST http://<<switch-ip>/vRest/ip-pools

{

  "name": "internal-ip-pool",
  "vlan": "111",
  "network": "172.16.21.0",
  "netmask": 24,
  "vnet-id": "a000024:1"

}
```

Request URL

```
http://<switch-ip>:80/vRest/ip-pools
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": "Pool created successfully"
  }
}
```

Response Code

201

Response Header

```
{
  "Content-Type": "application/json"
}
```

Display IP Pools

To display all IP Pools configured on the server-switch, use the following REST API:

```
GET http://<switch-ip>/vRest/ip-pools
```

```
{
  "ip-pool-show": {
    "ip-pool": [
      {
        "name": "",
        "scope": "",
        "vlan": "short",
        "network": "",
        "netmask": 0,
        "vnet-id": "",
        "start-ip": "",
        "end-ip": ""
      }
    ]
  },
  "result": {
    "status": "",
    "code": 0,
    "message": ""
  }
}
```

Request URL

```
GET http://<switch-ip>:80/vRest/ip-pools
```

Response Body

```
      {
        "name": "internal-net",
        "vnet-id": "a000024:1",
        "scope": "local",
        "vlan": 111,
        "start-ip": "172.16.21.1",
        "end-ip": "172.16.21.254",
        "network": "172.16.21.0",
        "netmask": 24
      }
    ],
    "result": {
      "status": "Success",
      "code": 0,
      "message": ""
    }
  }
```

Response Code

200

Response Headers

```
  {
    "Content-Type": "application/json"
  }
```

Changing the IP Pool Parameters

To change the VLAN of an IP Pool, use the following REST API:

```
  PUT http://<switch-ip>/ip-pools/{name}

  {
          "name": "",
          "vlan": "short",
          "network": "",
          "netmask": 0,
          "vnet-id": "",
          "start-ip": "",
          "end-ip": ""


    }
```

However, since you are only changing the VLAN assigned to the IP Pool, remove the rest of the parameters and the comma after "`short`". Then execute the API. If you don't remove the final comma, you may see a Response Body error such as "malformed content".

Request URL

```
http://<switch-ip>:80/vRest/ip-pools/internal-net
```

```
{
    “name”: “internal-net”,
    “vlan”: “111”
}
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Code

200

Response Headers

```
{
  "Content-Type": "application/json"
}
```

Adding DHCP Services

Now, the VNET, internal-network, needs DHCP added so that when interfaces are added to the VNET, the interfaces obtain an IP address from the DHCP services.

To create the DHCP service, **internal-dhcp**, use the following REST API:

```
POST http://<switch-ip>/dhcps
```

```
{

  "name": "",
  "vnet": "",
  "vnet-service": false,
  "storage-pool": "",
  "initial-ip-pool": "",
  "max-leasetime": 0,
  "default-leasetime": 0,
  "pxe-boot": "",
  "pxe-default-menu": "",
  "pxe-menu-timeout": 0
}
```

Request URL

```
http://<switch-ip>:80/vRest/dhcps
```

```
{

  "name": "internal-dhcp",
  "vnet": "internal-net",
  "vnet-service": false,
  "storage-pool": "",
  "initial-ip-pool": "internal-ip-pool"
}
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Code

201

Response Header

```
{
  "Content-Type": "application/json"
}
```

CLI Show Output

CLI server-switch > **CLI (network-admin@pubdev02) > dhcp-show name internal-dhcp format all layout vertical**

```
switch:              pubdev02
id:                  a000024:4
name:                internal-dhcp
type:                dhcp
scope:               local
vnet:                internal-net
vnet-service:        shared
state:               enabled
location:            pubdev02
storage-pool:        rpool
gateway:             ::
template:            no
max-leasetime:       3600
default-leasetime:   3600
pxe-boot:            disabled
pxe-default-menu:
pxe-menu-timeout:    0
kickstarts-share:    /net/pubdev02/nvOS/internal-net-mgr/kickstarts
```

Adding the Gateway IP to DHCP

You can add the IP address that you want to use the gateway IP address using the following REST API:

```
PUT http://<switch-ip>/dhcps/internal-dhcp/pools/internal-dhcp-pool

{
  "gateway-ip": "",
  "ddns-domain": "",
  "dns-ip": "",
  "dns2-ip": "",
  "ddns-domain-rev": "",
  "ntp-server-ip": ""

}
```

Request URL

```
http://<switch-ip>:80/vRest/dhcps/internal-dhcp/pools/internal-dhcp-pool

{
   "dhcp-pool-name": "internal-ip-pool",
   "gateway-ip": "172.16.21.1"
}
```

Response Body

```
{
  "result": {
    "status": "Success",
    "code": 0,
    "message": ""
  }
}
```

Response Header

```
200
```

Response Headers

```
{
  "Content-Type": "application/json"
}
```

CLI Show Output

```
CLI server-switch > dhcp-show format all layout vertical

 switch:          pubdev02
 dhcp-name:       internal-dhcp
 dhcp-pool-name:  internal-ip-pool
 dhcp-ip-pool:    internal-ip-pool
 start-ip:        172.16.21.1
 end-ip:          172.16.21.254
 gateway-ip:      172.16.21.1
 ddns-domain:
 dns-ip:          ::
 dns2-ip:         ::
 ddns-domain-rev:
 kickstarts-url:  http://172.16.21.1/kickstarts/
 ntp-server-ip:   ::
```

Adding DHCP to a VNET Interface

Before you can add DHCP to a VNET interface, you should display the VNET manager show output to obtain the VNET manager name. If you don't specifically configure it, then it defaults to *vnet-name*-manager.

To add DHCP to a VNET interface, use the following REST API:

```
PUT http://<switch-ip>/vnet-managers/{vnet-manager-name}/interfaces

{
  "ip": "",
  "netmask": 0,
  "exclusive": false,
  "assignment": "",
  "vlan": "short",
  "vxlan": 0,
  "if": "",
  "alias-on": "",
  "nic-config": false,
  "vrrp-id": 0,
  "vrrp-primary": "",
  "vrrp-priority": "",
  "l3-port": 0,
  "secondary-macs": ""
}
```

Request URL

```
http://<switch-ip>:80/vRest/vnet-managers/internal-net-mgr/interfaces

{
  "assignment": "dhcp",
  "vlan": "111",
  "if": "mgmt"

}
```

Pluribus Networks is simplifying the Software-Defined Data Center with its simple, dynamic and secure Adaptive Cloud Fabric architecture, enabling organizations to build scalable private and public clouds that improve service velocity, performance, and reliability.

The company's innovative Netvisor software virtualizes open networking hardware to build a holistic, distributed network that is more intelligent, automated and resilient.

The company's Insight Analytics platform leverages embedded telemetry and other data sources to enable pervasive visibility across the network to reveal network and application performance that speeds troubleshooting and improves operational and security intelligence. Pluribus Networks has received venture funding from Temasek Holdings, NEA, Menlo Ventures, and AME Cloud Ventures.

Pluribus Networks is headquartered in Santa Clara, California, with development and support centers in Bangalore, India; Phoenix, AZ; Dallas, TX; and Hong Kong, PRC.

For additional information contact Pluribus Networks at info@pluribusnetworks.com, or visit www.pluribusnetworks.com.

Follow us on Twitter @pluribusnet.

PLURIBUS NETWORKS
Corporate and Sales Headquarters
Pluribus Networks, Inc.
5201 Great America Parkway, Suite 422
Santa Clara, CA 95054 USA
U.S. and Canada Toll-Free: 1-855-GET-VNET
www.pluribusnetworks.com