Pluribus Networks

Netvisor® ONE

REST API Guide

Version 5.2.1

June 2020

Table of Contents

Legal Notice

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 AND 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.

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 schema 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.

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.

vRestApi implements the REST API Web service and interacts with Netvisor ONE.

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

REST API Design

REST API Clients

The REST API does not have a client-side library. 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. Typically, REST APIs are implemented within Web frameworks and 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.

Swagger provides the documentation, and you use the Swagger JavaScript client in a Web browser.

Swagger, designed for REST API documentation and the JavaScript client, provides a convenient and quick way to display REST URIs and JSON, as well as a way to test a URI.

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

See cUrl examples for more information.

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/asdasd (where the name of the vFlow is asdasd)

URL = http://10.110.0.56:80/vRest/vflows/name/asdasd

{"vflow":
{"name":"asdasd","id":"c000184:52","scope":"local","type":"vflow","hidden":
false,"burst-size":0,"precedence":2,"log-stats":true,"stats-
interval":60,"hw-stats":true,"enable":true,"table-name":"System-L1-L4-Tun-
1-0"}}{"result":{"api.switch-
name":"local","scope":"local","status":"Success","code":0,"message":""}}
```

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.

REST API Design (cont'd)

The key elements of a RESTful implementation are as follows:

- **Resources** – The first key element is the resource itself. You want to create a VLAN on a switch, and the URL of the switch is `http://<switch-ip>`. In order to create a VLAN using REST, you can issue the command `http://<switch-ip>/vREST/vlans`. This command creates the vlan per the settings in the Parameter body field.

- **Request Verbs** – 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/vlans`, the Web browser issues a `POST` verb because you want to configure a vlan. See VLAN example.

- **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

The following is a list of actions of 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** – Adds the supplied information to the specified object; returns a 404 Resource Not Found error if the object does not exist.
- **DELETE** – used to delete a configuration using the RESTful web service.

REST API Design (cont'd)

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.
- **401** (Unauthorized) – unsuccessful authentication with Netvisor ONE®.
- **403** (Forbidden) – authentication is successful however the command failed to execute. Details are provided.
- **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”,”code”:integer,”messag
e”:string}}}
```

The response payload may contain data, and always contains `result`.

REST API Design (cont'd)

Conventions

- The `data` consists of one or more resources of the same type.
- The `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`.
- The `result` may contain a code number or a message that provides further information about the back-end operation on Netvisor ONE, such as the next value or an error.

REST API clients should check both the `HTTP status code` returned in the response Status-Line and the `responseCode` returned in the `result`.

When the message in the status code response is "Success", the Response Code is either 200 or 201.

In the event of a "Failure" message in the status code, check for the specific Response Code to diagnose the problem. In some cases, data may contain a partial result.

REST API Switch Configuration Settings

Configuring REST API Access

Netvisor ONE® enables you to use REST API over HTTP and HTTPS to manage the switches in a fabric, in addition to using the CLI.

Though REST API access over HTTP is simpler to configure, Pluribus Networks recommends using HTTPS for security reasons.

The vREST web application that runs on the switch enables the REST API client to access the switch's resources.

Follow the steps below to configure REST API access over HTTP:

Enable the web service using the command: `admin-service-modify`.

```
CLI (network-admin@switch1) admin-service-modify if mgmt web
```

`admin-service-modify`	Modify services on the switch.
`if if-string`	Specify the administrative service interface. The options are mgmt or data.
Specify one or more of the following options:	
`ssh|no-ssh`	Specify if you want to connect to the switch using Secure Shell (SSH).
`nfs|no-nfs`	Specify if you want to use Network Files System (NFS) for the administrative service.
`web|no-web`	Specify if you want to enable web management. **Use this option to enable REST API access over HTTP.**
`web-ssl|no-web-ssl`	Specify if you want to use SSL and certificates for web services. **Use this option to enable REST API access over HTTPS.**
`web-ssl-port web-ssl-port-number`	Specify the web SSL port.
`web-port web-port-number`	Specify the port for web management.
`web-log|no-web-log`	Specify if you want to turn on or off web logging.
`snmp|no-snmp`	Specify if SNMP is allowed as a service.
`net-api|no-net-api`	Specify if APIs are allowed as a service.
`icmp|no-icmp`	Specify if Internet Control Message Protocol (ICMP) is allowed as a service.

REST API Switch Configuration Settings (cont'd)

Verify the configuration using the command: `admin-service-show`:

```
CLI (network-admin@switch1) admin-service-show

switch      if   ssh nfs web web-ssl web-ssl-port web-port snmp net-api
icmp
----------- ---- --- --- --- ------- ------------ -------- ---- -------
----
switch1     mgmt on  off on  on      443          80       on   off     on
switch1     data on  off on  off     443          80       on   off     on
```

To access the log details, enable the `web-log` parameter by using the command:

```
CLI (network-admin@switch1) > admin-service-modify if mgmt web-log
```

Warning: We recommend enabling `web-log` for debugging purposes and only as advised by **Pluribus Networks Technical Support** as log files can quickly consume available disk space.

If you wish to confirm `web_log` is enabled run the following command:

```
CLI (network-admin@udev-leo1) > admin-service-show format all
```

To disable the `web-log` run the following command:

```
CLI (network-admin@switch1) > admin-service-modify if mgmt no-web-log
```

Configuring REST API Access over HTTPS

To enable HTTPS communication between a REST API client and Netvisor vREST web service, you have two options:

1. You can generate a self-signed certificate using Netvisor CLI and use this certificate for the REST web service.

2. After creating a self-signed certificate using Netvisor CLI, create a certificate request, get the certificate request signed by a trusted Certificate Authority (CA), import the signed certificate and CA certificate into Netvisor ONE, and use the certificates for REST API web service.

REST API Switch Configuration Settings (cont'd)

Follow the steps below to create the certificates and deploy them:

Generate self-signed certificate (the private key and the certificate file, in PEM format) using the `web-cert-self-signed-create` command.

```
CLI (network-admin@switch1) > web-cert-self-signed-create
```

`web-cert-self-signed-create`	This command creates a self-signed certificate and deletes any existing certificates.
`country country-string`	Specify the contact address of the organization, starting with the country code.
`state state-string`	Specify the state or province.
`city` *`city-string`*	Specify the city.
`organization` *`organization-string`*	Specify the name of the organization.
`organizational-unit` *`organizational-unit-string`*	Specify the organizational unit.
`common-name` *`common-name-string`*	Specify the common name. The common name must precisely match the hostname where the certificate is installed.

For example:

```
CLI (network-admin@switch1) > web-cert-self-signed-create country US state
California city "Santa Clara" organization "Pluribus Networks Inc"
organizational-unit Engineering common-name switch1.pluribusnetworks.com
Successfully generated self-signed certificate.
```

This command generates the certificate request and saves the files internally.

Enable `web-ssl` by using the `admin-service-modify` command.

```
CLI (network-admin@switch1) admin-service-modify if data web-ssl
```

REST API Switch Configuration Settings (cont'd)

If you want to get the certificate signed by a trusted Certificate Authority (CA), generate a CSR from the self-signed certificate by using the command `web-cert-request-create`.

```
CLI (network-admin@switch1) > web-cert-request-create
Certificate signing request successfully generated
at /sftp/export/switch1.pluribusnetworks.com.csr
```

To view the CSR, use the command web-cert-request-show.

```
CLI (network-admin@switch1) > web-cert-request-show
```

`web-cert-request-show`	Displays the certificate signing request.
`cert-request` *`cert-request-string`*	Specify the name of the CSR.

For example:

```
CLI (network-switch1) > web-cert-request-show

cert-request
------------------------------------------------------------------
-----BEGIN CERTIFICATE REQUEST-----
MIICnDCCAYQCAQEwVzELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAkNBMQswCQYDVQQH
DAJTSjELMAkGA1UECgwCUE4xDTALBgNVBAsMBEVuZ2cxEjAQBgNVBAMMCWVxLWNv
bG8tMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALMmrZ8hvZ5J+FRs
Lo1sfVtwmmLEaxyhaxD/HNVdXSRhzbQDT20+qySfOudxtWGKyCsuCFFbgMUz7rgu
H1Xle8uwPSoxgTjLGq20sgBQIfNBT5UwDLDuzUUPzMEEjFb3/9Cg1VWju2t1KPim
Gqg3rcA3PCsMeCr/q+9Gz6gfLe6Rfx91yxTA44ZWsOWnvgDdXAPfHOLZ5zBWG8a3
ohgOwMLjy21ytDTA6aR1M9I12MkJwev3t0y6n/CLp6Zigp5wXiArPPnR9sZ+E7so
MqpEzz0rjFDfrNwNAGMzT3WPcmlYRjYrUJ0QsOEQ+O1uHJaNbw1pJEmK2jm97kbk
/HvEFmMCAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4IBAQCnlgEwzoesbuiCYG7HZJN/
Rxm/NcznpvJXxdlTAdzSbTWWLswrZMyX6bQqUTWEb3qvVccD4tIZShyIGiR0CpCD
22m8LD4+e6/FA6NijjanHkKsRW9Z7ka97TFpsUaH27sUTtfFDDkDImwRIGfns+nu
kTRNMuNiyC/+uHovsvCxS8is3OasQtS1lkG28sZgxisvP17qmfjlb9fQC3pcvR4t
K8GciPMUfgcIA5qLDmCZAg1A6JBMb/UHtUuEnztLrLz4qjWqJJK3pWvdLWZcKDEz
C0t5Dre9ByJ2RT75GdUq2c16xYBGAwZNCzjdhParyBnvn00Mwb6PpPmLGcBQiRNn
-----END CERTIFICATE REQUEST-----
```

REST API Switch Configuration Settings (cont'd)

Send the CSR to your trusted CA. You can copy the `web-cert-request-show` output and send it to the CA for signing the certificate.

You can also connect to the switch by using `SFTP` and copy the certificate file from `/sftp/export` location and send it to the CA.

If disabled, use the command `admin-sftp-modify` enable to enable SFTP.

In return, the CA provides the server certificate of your switch signed using the intermediate key.

Upload the signed certificate, the CA root certificate, and the intermediate CA certificate (if an intermediate CA signs the certificate) to `/sftp/import` directory on the switch using `SFTP`.

For example:

```
$ sftp sftp@switch1

Password:pluribus_password

sftp> cd /sftp/import

sftp> put server-cert.pem
```

Import the signed server certificate, CA root certificate, and the intermediate certificate (if available) onto the switch using the web-cert-import command:

```
CLI (network-admin@switch1) > web-cert-import
```

`web-cert-import`	This command imports certificates from `/sftp/import` directory.
`file-ca` *`file-ca-string`*	Specify the name of the CA certificate file.
`file-server` *`file-server-string`*	Specify the name of server certificate file (signed by CA).
`file-inter` *`file-inter-string`*	Specify the name of intermediate CA certificate file.

```
CLI (network-admin@switch1) > web-cert-import file-ca ca.pem file-server
server-cert.pem file-inter intermediate.pem
Successfully imported certificates.
```

After the import is successful, enable `web-ssl` using the `admin-service-modify` command.

```
CLI (network-admin@switch) > admin-service-modify if data web-ssl
```

REST API Switch Configuration Settings (cont'd)

Related Commands

- `web-cert-clear`

Use this command to delete previously generated certificates.

For example:

```
CLI (network-admin@switch1) > web-cert-clear
Successfully deleted all certificate files.
```

- `web-cert-info-show`

Use this command to display web certificate information.

```
CLI (network-admin@switch1) web-cert-info-show
```

`web-cert-info-show`	Displays the web certificate information.
Specify any of the following options:	
`cert-type ca|intermediate|server`	Specify the one among the options as the certificate type.
`subject` *`subject-string`*	Specify the the subject of the certificate.
`issuer` *`issuer-string`*	Specify the issuer of the certificate.
`serial-number` *`serial-number`*	Specify the serial number of the certificate.
`valid-from` *`valid-from-string`*	Specify the time from which the certificate is valid.
`valid-to` *`valid-to-string`*	Specify the time at which the certificate expires and is no longer valid.

REST API Switch Configuration Settings (cont'd)

For example:

```
CLI (network-admin@switch1) web-cert-info-show

switch:         switch1
cert-type:      ca
subject:        /C=US/ST=CA/L=SJ/O=PN/OU=Engg/CN=switch1
issuer:         /C=US/ST=CA/L=SJ/O=PN/OU=Engg/CN=switch1
serial-number: 1
valid-from:     May  7 18:16:10 2019 GMT
valid-to:       May  6 18:16:10 2020 GMT
----------------------------------------
switch:         switch1
cert-type:      server
subject:        /C=US/ST=CA/L=SJ/O=PN/OU=Engg/CN=switch1
issuer:         /C=US/ST=CA/L=SJ/O=PN/OU=Engg/CN=switch1
```

REST API Switch Configuration Settings (cont'd)

Using cURL to Implement SSL Certs

Use `cURL` to automate the upload of the CA root, CA intermediate and signed switch certificates.

Run the following command for each of the PEM formatted certificates:

```
awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' <file-name>.pem
```

Example

```
$ awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' /tmp/server-cert.pem.bkp

-----BEGIN CERTIFICATE-----
\nMIIDHDCCAgQCAQEwDQYJKoZIhvcNAQELBQAwVDELMAkGA1UEBhMCSU4xCzAJBgNV
\nBAgMAktBMQwwCgYDVQQHDANCTFIxCzAJBgNVBAoMAlBOMQwwCgYDVQQLDANFTkcx
\nDzANBgNVBAMMBlNQSU5FMTAeFw0yMDA1MDQxODM4NTZaFw0yMTA1MDQxODM4NTZa
\nMFQxCzAJBgNVBAYTAklOMQswCQYDVQQIDAJLQTEMMAoGA1UEBwwDQkxSMQswCQYD
\nVQQKDAJQTjEMMAoGA1UECwwDRU5HMQ8wDQYDVQQDDAZTUElORTEwggEiMA0GCSqG
\nSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCot16ddH0LNHOrWZt63FHYiArVXYIYbCDh
\nWCY6MX3suoXYvKstvRgJkUe/6G5As+vYtwRi2bDqdDsgTC5+Qo4SnjrdTcTM98F+
\n0Qzqv02c+dbzk5GclUkljqq0PHGXRPGOMhou8B/6LI9Hg5XkG+FSfaDGQTM39uj2
\nzzvrMOFn96gzpTBoh40sMoIpnKQLrWeGjlNxaBxhM342c1jn1CVmXss/uHMQeang
\nsVhPTynikyxIrDwl9gh/2X1EwzVzpAnUBTUZvJ9rgrceC9GcuGmiPZgxxSruNb0w
\nK8xsyH8/hLwhK4Axgu3a+lfmmKFmSWjywmcxlmQl+jwiMPA/Ty55AgMBAAEwDQYJ
\nKoZIhvcNAQELBQADggEBAEG0D/2FcNU6Z6w/6eKbyH855kHSrJyqeU8eoCW9rnOb
\nqdnAsFX3aYwiUCjzSFXpWA3bRr3L7X0Y01x7VSvwITuDvwO43llK29rQfrSvoPiw
\nf7fhU7bszlUc2GAumU9OEdYBnSI1DzfBawUcPmbDmm+ci27k0po53KDWTbxkBIZR
\n2Oh25LXkmq8ZBzE4vgS+mAw436nToazB1/vDTMWoBuLVzOUlU8cdcjJUnJBevTbX
\nThP691sHVMED8B8Fhl08BzIJmQQ9qp1tjplFq1Ea9oEFnT5U5gKvJYy48qEPlW+r
\nhRIHysvZXF/dghtrLXDMSBWLlLofUsDQsh+qLxpo1+k=
\n-----END CERTIFICATE-----\n
```

Warning: Failure to use the escape character syntax of \n, as highlighted in red in the examples shown, results in the script failing, and the installation of the certificates to fail.

Note: Certificate examples on this page are displayed line-wrapped for purposes of documentation clarity only.

REST API Switch Configuration Settings (cont'd)

Copy the output into the json payload.

```
$ curl -u network-admin:pluribus_password http://10.100.64.5/vRest/web-
certs/upload -H "content-type:application/json" -v -X POST -d '{"cert-
ca":"-----BEGIN CERTIFICATE-----
\nMIIDHDCCAgQCAQEwDQYJKoZIhvcNAQELBQAwVDELMAkGA1UEBhMCSU4xCzAJBgNV
\nBAgMAktBMQwwCgYDVQQHDANCTFIxCzAJBgNVBAoMAlBOMQwwCgYDVQQLDANFTkcx
\nDzANBgNVBAMMBlNQSU5FMTAeFw0yMDA1MDQxODM4NTZaFw0yMTA1MDQxODM4NTZa
\nMFQxCzAJBgNVBAYTAklOMQswCQYDVQQIDAJLQTEMMAoGA1UEBwwDQkxSMQswCQYD
\nVQQKDAJQTjEMMAoGA1UECwwDRU5HMQ8wDQYDVQQDDAZTUElORTEwggEiMA0GCSqG
\nSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCot16ddH0LNHOrWZt63FHYiArVXYIYbCDh
\nWCY6MX3suoXYvKstvRgJkUe/6G5As+vYtwRi2bDqdDsgTC5+Qo4SnjrdTcTM98F+
\n0Qzqv02c+dbzk5GclUkljqq0PHGXRPGOMhou8B/6LI9Hg5XkG+FSfaDGQTM39uj2
\nzzvrMOFn96gzpTBoh40sMoIpnKQLrWeGjlNxaBxhM342c1jn1CVmXss/uHMQeang
\nsVhPTynikyxIrDwl9gh/2X1EwzVzpAnUBTUZvJ9rgrceC9GcuGmiPZgxxSruNb0w
\nK8xsyH8/hLwhK4Axgu3a+lfmmKFmSWjywmcxlmQl+jwiMPA/Ty55AgMBAAEwDQYJ
\nKoZIhvcNAQELBQADggEBAEG0D/2FcNU6Z6w/6eKbyH855kHSrJyqeU8eoCW9rnOb
\nqdnAsFX3aYwiUCjzSFXpWA3bRr3L7X0Y01x7VSvwITuDvwO43llK29rQfrSvoPiw
\nf7fhU7bszlUc2GAumU9OEdYBnSI1DzfBawUcPmbDmm+ci27k0po53KDWTbxkBIZR
\n2Oh25LXkmq8ZBzE4vgS+mAw436nToazB1/vDTMWoBuLVzOUlU8cdcjJUnJBevTbX
\nThP691sHVMED8B8Fhl08BzIJmQQ9qp1tjplFq1Ea9oEFnT5U5gKvJYy48qEPlW+r
\nhRIHysvZXF/dghtrLXDMSBWLlLofUsDQsh+qLxpo1+k=
\n-----END CERTIFICATE-----\n",
"cert-server":"-----BEGIN CERTIFICATE-----
\nMIIDHDCCAgQCAQEwDQYJKoZIhvcNAQELBQAwVDELMAkGA1UEBhMCSU4xCzAJBgNV
\nBAgMAktBMQwwCgYDVQQHDANCTFIxCzAJBgNVBAoMAlBOMQwwCgYDVQQLDANFTkcx
\nDzANBgNVBAMMBlNQSU5FMTAeFw0yMDA1MDQxODM4NTZaFw0yMTA1MDQxODM4NTZa
\nMFQxCzAJBgNVBAYTAklOMQswCQYDVQQIDAJLQTEMMAoGA1UEBwwDQkxSMQswCQYD
\nVQQKDAJQTjEMMAoGA1UECwwDRU5HMQ8wDQYDVQQDDAZTUElORTEwggEiMA0GCSqG
\nSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCot16ddH0LNHOrWZt63FHYiArVXYIYbCDh
\nWCY6MX3suoXYvKstvRgJkUe/6G5As+vYtwRi2bDqdDsgTC5+Qo4SnjrdTcTM98F+
\n0Qzqv02c+dbzk5GclUkljqq0PHGXRPGOMhou8B/6LI9Hg5XkG+FSfaDGQTM39uj2
\nzzvrMOFn96gzpTBoh40sMoIpnKQLrWeGjlNxaBxhM342c1jn1CVmXss/uHMQeang
\nsVhPTynikyxIrDwl9gh/2X1EwzVzpAnUBTUZvJ9rgrceC9GcuGmiPZgxxSruNb0w
\nK8xsyH8/hLwhK4Axgu3a+lfmmKFmSWjywmcxlmQl+jwiMPA/Ty55AgMBAAEwDQYJ
\nKoZIhvcNAQELBQADggEBAEG0D/2FcNU6Z6w/6eKbyH855kHSrJyqeU8eoCW9rnOb
\nqdnAsFX3aYwiUCjzSFXpWA3bRr3L7X0Y01x7VSvwITuDvwO43llK29rQfrSvoPiw
\nf7fhU7bszlUc2GAumU9OEdYBnSI1DzfBawUcPmbDmm+ci27k0po53KDWTbxkBIZR
\n2Oh25LXkmq8ZBzE4vgS+mAw436nToazB1/vDTMWoBuLVzOUlU8cdcjJUnJBevTbX
\nThP691sHVMED8B8Fhl08BzIJmQQ9qp1tjplFq1Ea9oEFnT5U5gKvJYy48qEPlW+r
\nhRIHysvZXF/dghtrLXDMSBWLlLofUsDQsh+qLxpo1+k=
\n-----END CERTIFICATE-----\n"}'
```

REST API Switch Configuration Settings (cont'd)

Note: Unnecessary use of -X or --request, POST is already inferred.

```
*   Trying 10.100.64.5...
* TCP_NODELAY set
* Connected to 10.100.64.5 (10.100.64.5) port 80 (#0)
* Server auth using Basic with user 'network-admin'
> POST /vRest/web-certs/upload HTTP/1.1
> Host: 10.100.64.5
> Authorization: Basic bmV0d29yay1hZG1pbjp0ZXN0MTIz
> User-Agent: curl/7.54.0
> Accept: */*
> content-type:application/json
> Content-Length: 2348
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: GET, POST, DELETE, PUT
< Set-Cookie: JSESSIONID=C52C3170DEEAC8E4996FF428D152BF25; Path=/vRest/;
HttpOnly
< Date: Tue, 05 May 2020 19:34:05 GMT
< Content-Type: application/json
< Content-Length: 162
<
* Connection #0 to host 10.100.64.5 left intact
{"result":{"status":"Success","result":[{"api.switch-
name":"local","scope":"local","status":"Success","code":0,"message":"Succes
sfully uploaded certificates."}]}}
```

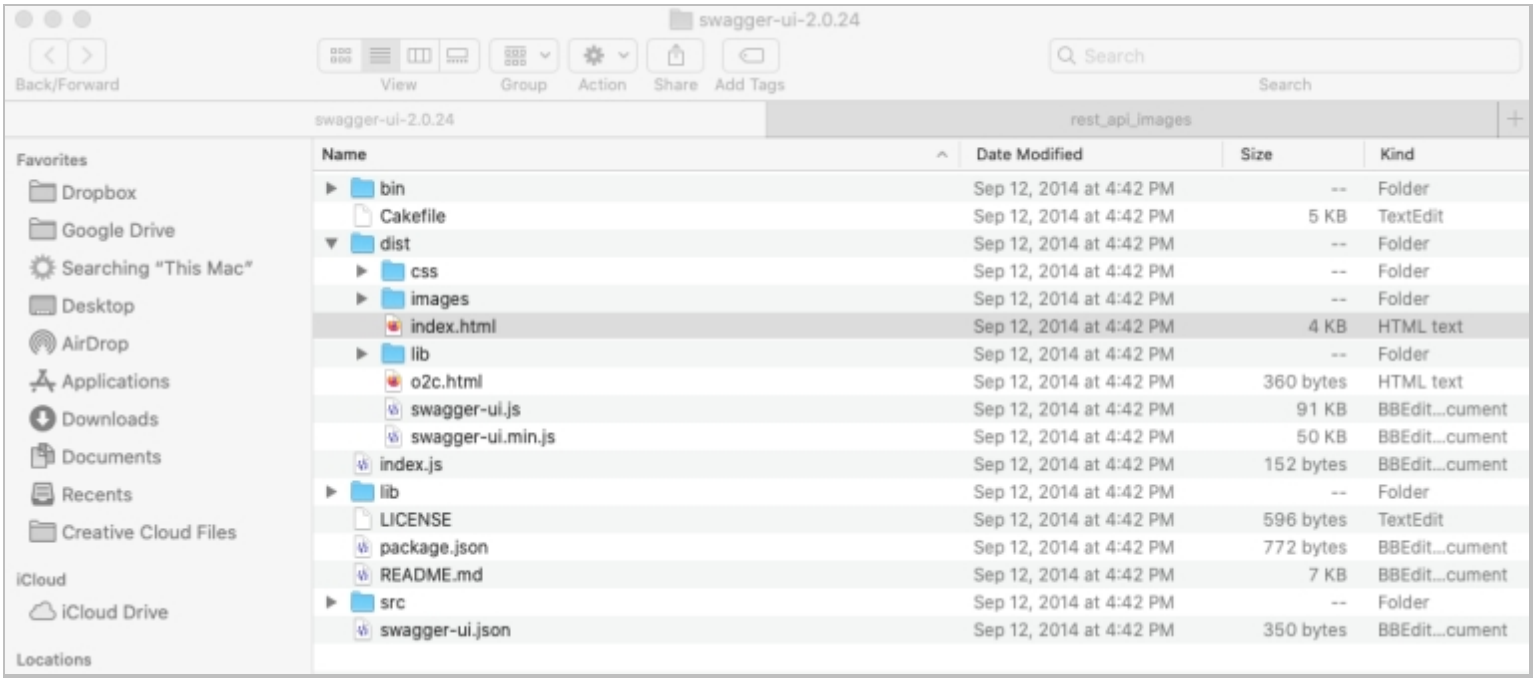
Setting Up Swagger

The REST API uses the Swagger-UI for documentation.

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

Swagger is a powerful tool for visualizing and testing REST APIs and requires manual steps to get everything working.

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



Setting Up Swagger (cont'd)

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
bmV0d29yay1hZG1pbjpwbHVyaWJ1c19wYXNzd29yZA==", "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");
```

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

Setting Up Swagger (cont'd)

index.html

~/Desktop/swagger-ui-2.0.24/dist/index.html master [anonymous]

```
<!DOCTYPE html>
<html>
<head>
  <title>Swagger UI</title>
  <link href='//fonts.googleapis.com/css?family=Droid+Sans:400,700' rel='stylesheet' type='text/css'/>
  <link href='css/reset.css' media='screen' rel='stylesheet' type='text/css'/>
  <link href='css/screen.css' media='screen' rel='stylesheet' type='text/css'/>
  <link href='css/reset.css' media='print' rel='stylesheet' type='text/css'/>
  <link href='css/screen.css' media='print' rel='stylesheet' type='text/css'/>
  <script type="text/javascript" src="lib/shred.bundle.js"></script>
  <script src='lib/jquery-1.8.0.min.js' type='text/javascript'></script>
  <script src='lib/jquery.slideto.min.js' type='text/javascript'></script>
  <script src='lib/jquery.wiggle.min.js' type='text/javascript'></script>
  <script src='lib/jquery.ba-bbq.min.js' type='text/javascript'></script>
  <script src='lib/handlebars-1.0.0.js' type='text/javascript'></script>
  <script src='lib/underscore-min.js' type='text/javascript'></script>
  <script src='lib/backbone-min.js' type='text/javascript'></script>
  <script src='lib/swagger.js' type='text/javascript'></script>
  <script src='swagger-ui.js' type='text/javascript'></script>
  <script src='lib/highlight.7.3.pack.js' type='text/javascript'></script>

  <!-- enabling this will enable oauth2 implicit scope support -->
  <script src='lib/swagger-oauth.js' type='text/javascript'></script>

  <script type="text/javascript">
    $(function () {
      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");
```

index.html

~/Desktop/swagger-ui-2.0.24/dist/index.html master [anonymous]

```
<!DOCTYPE html>
<html>
<head>
  <title>Swagger UI</title>
  <link href='//fonts.googleapis.com/css?family=Droid+Sans:400,700' rel='stylesheet' type='text/css'/>
  <link href='css/reset.css' media='screen' rel='stylesheet' type='text/css'/>
  <link href='css/screen.css' media='screen' rel='stylesheet' type='text/css'/>
  <link href='css/reset.css' media='print' rel='stylesheet' type='text/css'/>
  <link href='css/screen.css' media='print' rel='stylesheet' type='text/css'/>
  <script type="text/javascript" src="lib/shred.bundle.js"></script>
  <script src='lib/jquery-1.8.0.min.js' type='text/javascript'></script>
  <script src='lib/jquery.slideto.min.js' type='text/javascript'></script>
  <script src='lib/jquery.wiggle.min.js' type='text/javascript'></script>
  <script src='lib/jquery.ba-bbq.min.js' type='text/javascript'></script>
  <script src='lib/handlebars-1.0.0.js' type='text/javascript'></script>
  <script src='lib/underscore-min.js' type='text/javascript'></script>
  <script src='lib/backbone-min.js' type='text/javascript'></script>
  <script src='lib/swagger.js' type='text/javascript'></script>
  <script src='swagger-ui.js' type='text/javascript'></script>
  <script src='lib/highlight.7.3.pack.js' type='text/javascript'></script>

  <!-- enabling this will enable oauth2 implicit scope support -->
  <script src='lib/swagger-oauth.js' type='text/javascript'></script>

  <script type="text/javascript">
    $(function () { window.authorizations.add("authorization", new ApiKeyAuthorization("authorization", "Basic [redacted]", "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){
```

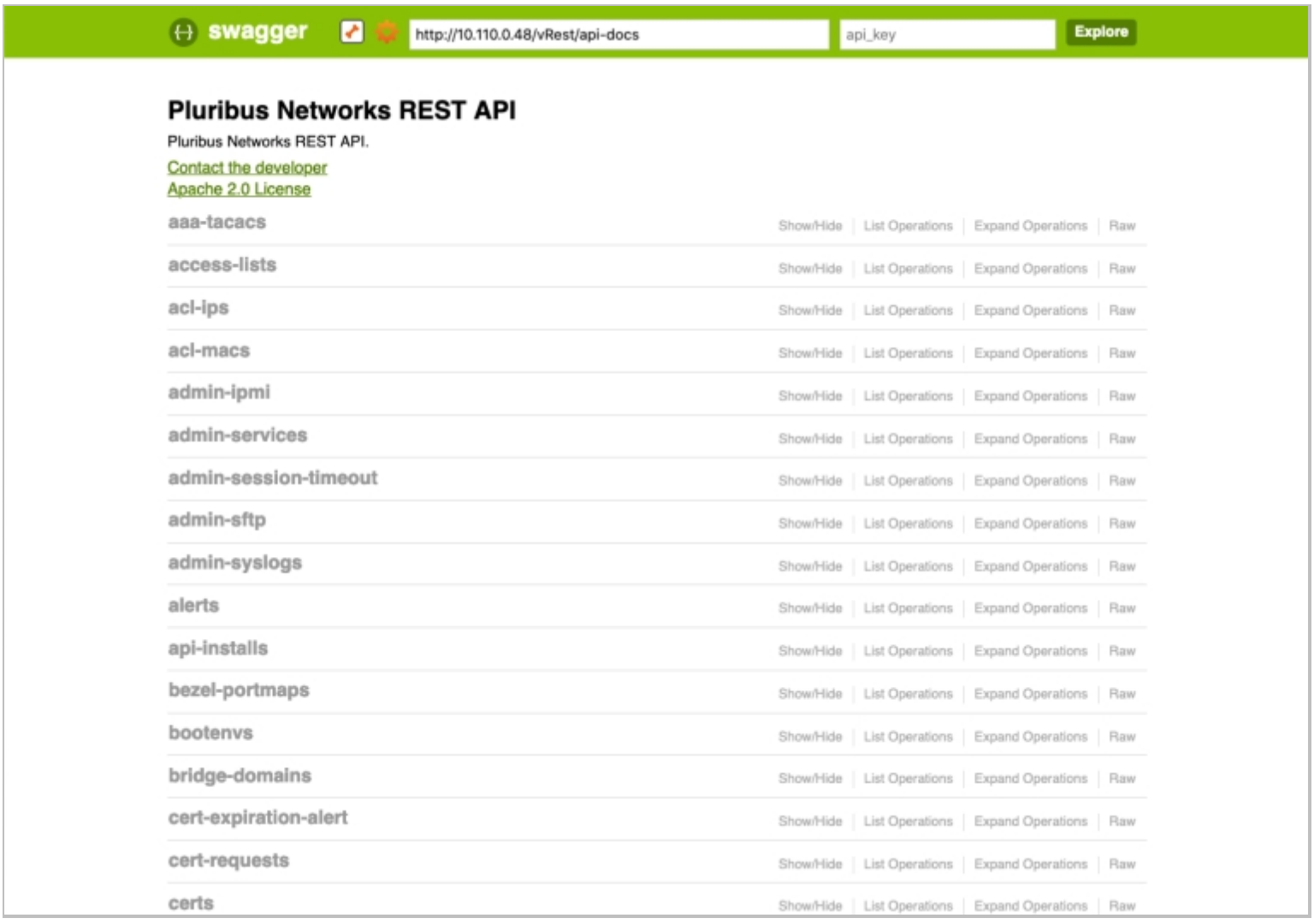
Setting Up Swagger (cont'd)

Examples

- This is the base 64 coded credential for `network-admin:pluribus_password` - `bmV0d29yay1hZG1pbjpwbHVyaWJ1c19wYXNzd29yZA==`
- This is the base 64 coded credential for `network-admin:p1zz@2015` - `bmV0d29yay1hZG1pbjpwMXp6QDIwMTU=`

To use another credential, use a base-64 encoding application such as https://www.base64encode.org/ to encode your `username:password` and substitute it in the above line.

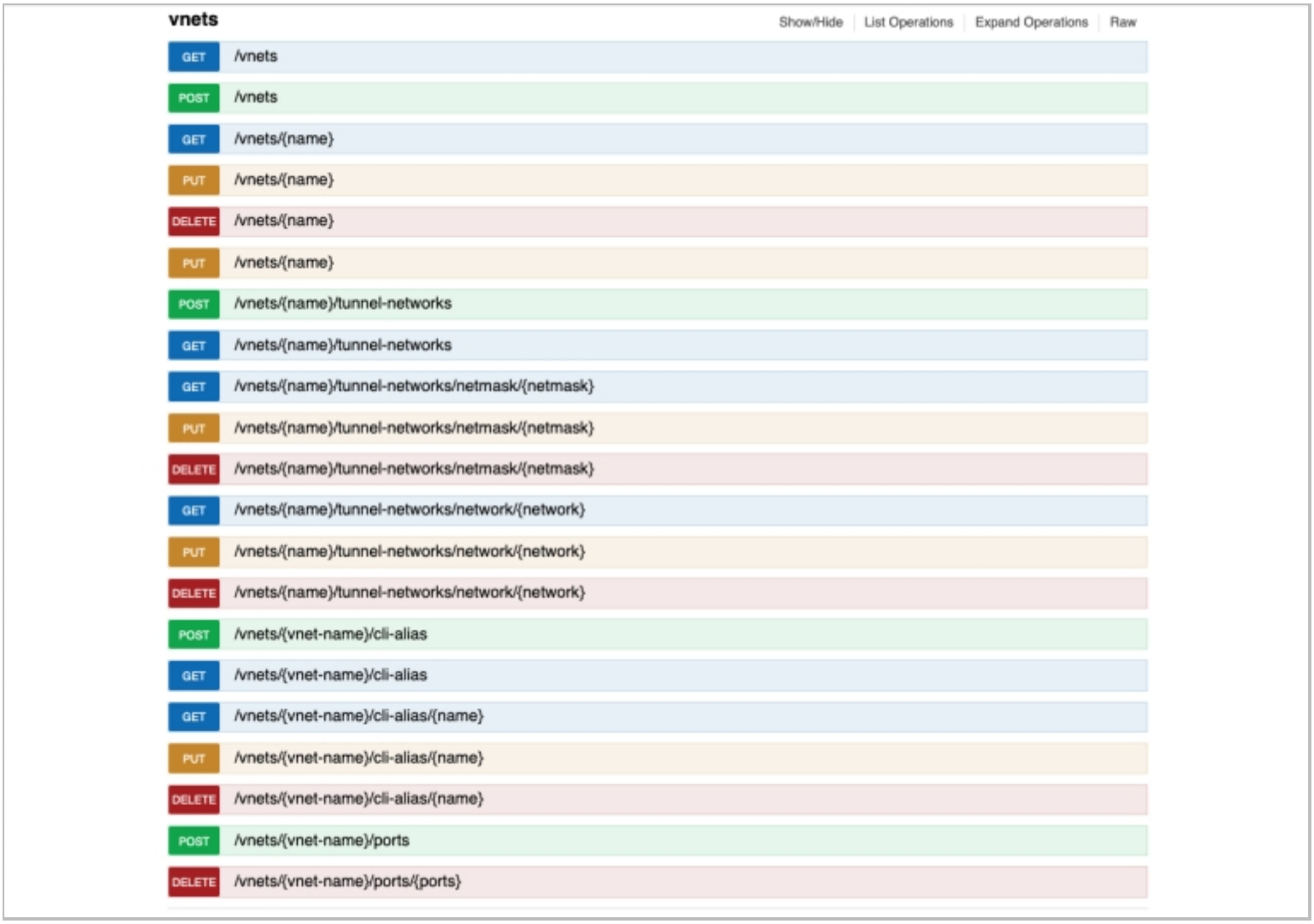
5. To access the Swagger REST API documentation:
   a) Load the file, `swagger-ui-2.0.24/dist/index.html`, into your browser.
   b) Paste the URL `http://<your-switch-mgmt-ip>/vRest/api-docs` into the Swagger URL field. To use secure HTTPS access refer to the REST API Switch Configurations Settings section for more information.
   c) Click **Explore**.



Setting Up Swagger (cont'd)

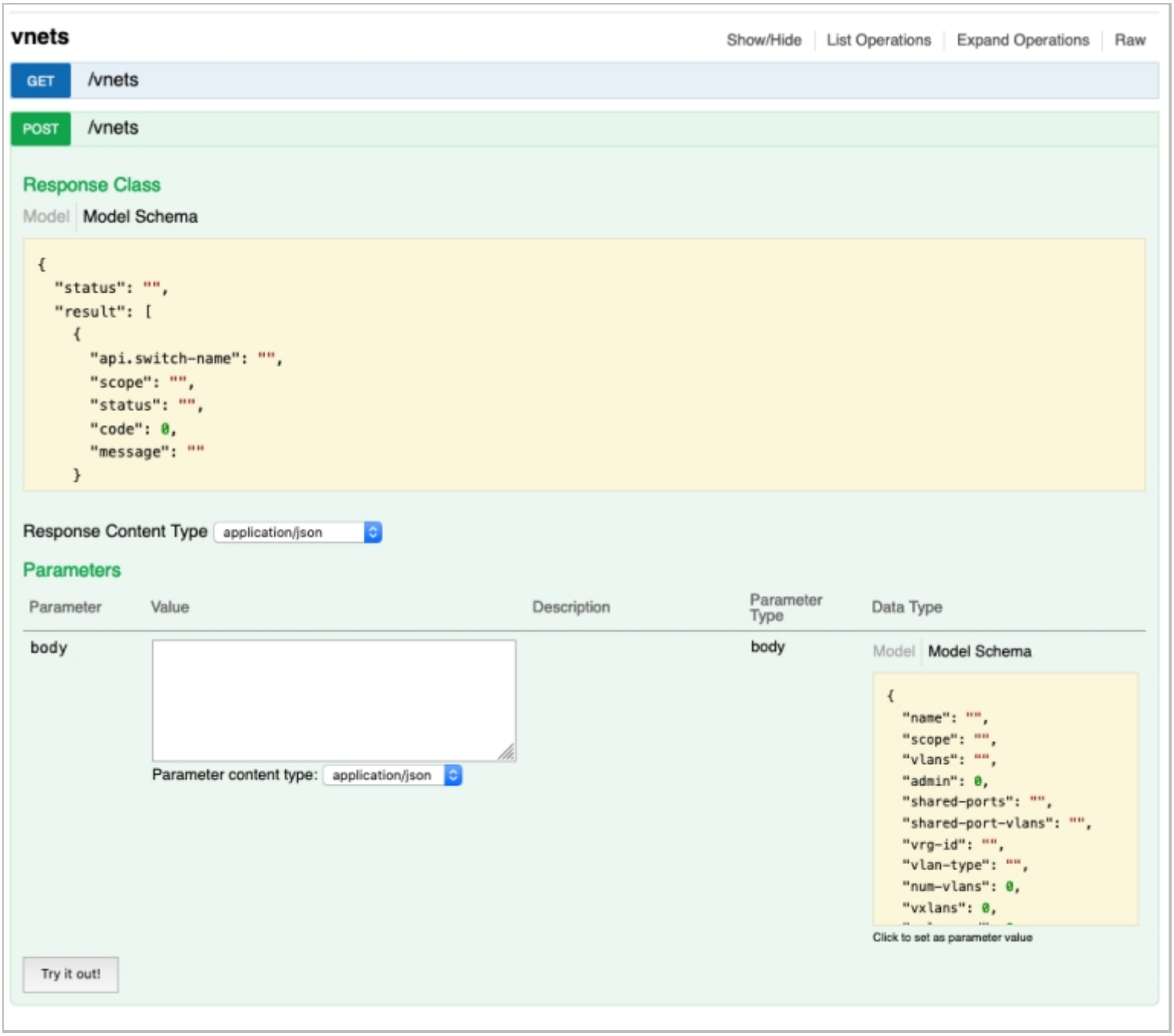
You can now browse the various APIs such as **vnets**.

> **Note:** Though the Swagger UI displays the list of API commands, it may take several minutes before you can perform REST API operations.



Setting Up Swagger (cont'd)

Test them using the Swagger “Try it out!” feature.



Setting Up Swagger (cont'd)

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`.

The following is the corresponding code snippet for Swagger-ui using: **network-admin:pluribus_admin** (example) credentials:

```
window.authorizations.add("authorization", new
ApiKeyAuthorization("authorization", "Basic
bmV0d29yay1hZG1pbjpwbHVyaWJ1c19hZG1pbg==", "header"))
```

REST API Examples

In general, you should not enclose numerical values such as port numbers 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 Configuration Guide
- Pluribus Networks Netvisor ONE Command Reference A-O
- Pluribus Networks Netvisor ONE Command Reference P-Z

available here to identify the parameters used by each command.

Using Swagger and REST API Examples

You can try out the following 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.

REST API Examples (cont'd)

Example 1: Get a **vFlow** identified by name:

```
GET /vflows/name/asdasd (where the name of the vFlow is asdasd)
```

Request URL

```
http://10.110.0.56:80/vRest/vflows/name/asdasd
```

Response Body

```
{"vflow":
{"name":"asdasd","id":"c000184:52","scope":"local","type":"vflow","hidden":
false,"burst-size":0,"precedence":2,"log-stats":true,"stats-
interval":60,"hw-stats":true,"enable":true,"table-name":"System-L1-L4-Tun-
1-0"}}
{"result":{"api.switch-
name":"local","scope":"local","status":"Success","code":0,"message":""}}
```

Response Code

```
200
```

Response Headers

```
{
 "Content-Type": "application/x-ndjson"
}
```

REST API Examples (cont'd)

Example 2: Get a **vFlow** identified by **ID**:

`GET /vflows/id/ce46fb` (where the id of the vflow is `c000184:52`)

Request URL

```
http://10.110.0.56:80/vRest/vflows/id/c000184%3A52
```

Response Body

```
{"vflow":
{"name":"asdasd","id":"c000184:52","scope":"local","type":"vflow","hidden":
false,"burst-size":0,"precedence":2,"log-stats":true,"stats-
interval":60,"hw-stats":true,"enable":true,"table-name":"System-L1-L4-Tun-
1-0"}}{"result":{"api.switch-
name":"local","scope":"local","status":"Success","code":0,"message":""}}
```

Response Code

```
200
```

Response Headers

```
{
 "Content-Type": "application/x-ndjson"
}
```

REST API Examples (cont'd)

Example 3: Create a **vFlow**:

```
POST /vflows {
  "name": "techpubs",
  "scope":"local",
  "burst-size":0,
  "precedence":2,
  "ether-type":"ipv4",
  "src-port":22,
  "dst-port":67,
  "src-ip":"10.110.0.48",
  "dst-ip":"10.110.0.50",
  "proto":1,
  "action":"copy-to-cpu"
}
```

Note: The last line of the script should not contain a final comma. If you do not remove the comma, you may see a **Response Body** error such as: `"There was a problem parsing the JSON input. Please check the JSON syntax and verify that the field values are the correct type."`

Request URL

```
http://10.110.0.48:80/vRest/vflows
```

Response Body

```
{
  "result": {
    "status": "Success",
    "result": [
      {
        "api.switch-name": "local",
        "scope": "local",
        "status": "Success",
        "code": 0,
        "message": "" }]}}
```

Response Code

```
201
```

REST API Examples (cont'd)

Example 3 - Create a **vFlow (cont'd)**

Response Headers

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

REST API Examples (cont'd)

Example 4: Delete a **vFlow**.

`DELETE /vFlow` (where the name of the vFlow is `techpubs`)

```
{
  "status": "",
  "result": [
    {
      "api.switch-name": "",
      "scope": "",
      "status": "",
      "code": 0,
      "message": ""
    }
  ]
}
```

Request URL

```
http://10.110.0.48:80/vRest/vflows/name/techpubs
```

Response Body

```
{"result":{"api.switch-
name":"local","scope":"local","status":"Success","code":0,"message":""}}
```

Response Code

```
200
```

Response Headers

```
{
 "Content-Type": "application/x-ndjson"
}
```

REST API Examples (cont'd)

Example 5: 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` (where user1 is `network-admin`)

Request URL

```
http://10.110.0.48:80/vRest/users/network-admin/roles
```

Response Body

```
{
  "data": [
    {
      "role-id": "0:0"
    }
  ],
  "result": {
    "status": "Success",
    "result": [
      {
        "api.switch-name": "local",
        "scope": "local",
        "status": "Success",
        "code": 0,
        "message": ""
      }
    ]
  }
}
```

Response Code

```
200
```

Response Headers

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

REST API Examples (cont'd)

Example 5 - Obtaining **User Role** Information (cont'd)

2. `GET /roles`

Request URL

```
http://10.110.0.48:80/vRest/roles
```

Response Body


```
{
  "data": [
    {
      "id": "0:0",
      "name": "network-admin",
      "scope": "local",
      "vnet-id": "0:0",
      "access": "read-write",
      "running-config": true,
      "shell": true,
      "sudo": false,
      "group-id": 20000
    },
    {
      "id": "0:1",
      "name": "read-only-network-admin",
      "scope": "local",
      "vnet-id": "0:0",
      "access": "read-only",
      "running-config": false,
      "shell": false,
      "sudo": false,
      "group-id": 20001
    } ],
  "result": {
    "status": "Success",
    "result": [
      {
        "api.switch-name": "local",
        "scope": "local",
        "status": "Success",
        "code": 0,
        "message": "" }]}}
```

REST API Examples (cont'd)

Example 5 - Obtaining **User Role** Information (cont'd)

Response Code

```
200
```

Response Headers

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

3. Match **Role** from first result set to **ID** from the second result set.

REST API Examples (cont'd)

Example 6: Update a **User Role**:

PUT /roles/name (where name is `techpubs_security_role` and change the role to `read-write access`)

```
{
  "access": "read-write",
  "shell": false,
  "sudo": false,
  "running-config": false,
  "delete-from-users": false
}
```

Request URL

```
http://10.110.0.48:80/vRest/roles/techpubs_security_role
```

Response Body

```
{
  "result": {
    "status": "Success",
    "result": [
      {
        "api.switch-name": "local",
        "scope": "local",
        "status": "Success",
        "code": 0,
        "message": ""
      }
    ]
  }
}
```

Response Code

```
200
```

Response Headers

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

REST API Examples (cont'd)

Example 7: Create and review a **VLAN**:

To create the VLAN:

```
POST /vlans
```

```
{
  "scope": "local",
  "id": 1411,
  "description": "techpubs-1"
}
```

Request URL

```
http://10.110.0.48:80/vRest/vlans
```

Response Body

```
{
  "result": {
    "status": "Success",
    "result": [
      {
        "api.switch-name": "local",
        "scope": "local",
        "status": "Success",
        "code": 0,
        "message": "Vlans 1411 created"
      }
    ]
  }
}
```

Response Code

```
201
```

Response Headers

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

REST API Examples (cont'd)

Example 7. Create and review a **VLAN** (cont'd)

To review the VLAN:

```
GET /vlans/id/{id}(where id = 1411)
```


```
{
  "data": [
    {
      "api.switch-name": "",
      "vnet-id": "",
      "id": 0,
      "type": "",
      "scope": "",
      "active": false,
      "stats": false,
      "flags": "",
      "hw-vpn": 0,
      "hw-mcast-group": 0,
      "repl-vtep": "",
      "vrg-id": "",
      "send-ports": "",
      "active-edge-ports": "",
      "ports-specified": false,
      "hw-member-ports": "",
      "public-vlan": 0,
      "vxlan": 0,
      "auto-vxlan": false,
      "vxlan-mode": "",
      "replicators": "",
      "ports": "",
      "untagged-ports": "",
      "description": ""
    } ],
  "result": {
    "status": "",
    "result": [
      {
        "api.switch-name": "",
        "scope": "",
        "status": "",
        "code": 0,
        "message": "" }]}}
```

REST API Examples (cont'd)

Example 7. Create and review a **VLAN** (cont'd)

Parameters

```
id = 1411
```

Request URL

```
http://10.110.0.48:80/vRest/vlans/id/1411
```

Response Body


```
{
  "data": [
    {
      "id": 1411,
      "type": "public",
      "auto-vxlan": false,
      "hw-vpn": 0,
      "hw-mcast-group": 0,
      "repl-vtep": "0:0:0:0:0:0:0:0",
      "scope": "local",
      "description": "techpubs-1",
      "active": true,
      "stats": true,
      "vrg-id": "0:0",
      "ports":
"1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28
,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53
,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,272,273,274",
      "untagged-ports": "",
      "active-edge-ports": ""
    }
  ],
  "result": {
    "status": "Success",
    "result": [
      {
        "api.switch-name": "local",
        "scope": "local",
        "status": "Success",
        "code": 0,
        "message": "" }]}}
```

REST API Examples (cont'd)

Example 7. Create and review a **VLAN** (cont'd)

Response Code

```
200
```

Response Headers

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

REST API Examples (cont'd)

Using cURL with the REST API

To create a **VLAN**, use the vREST API:

```
$ curl -u network-admin:pluribus_password -H "Content-
Type:application/json" -X POST http://switch1/vRest/vlans -d '{"scope":
"local","id": 1111,"description": "hello world"}'
```

A successful execution of the above cURL command returns the result:

```
{"result":{"status":"Success","result":[{"api.switch-
name":"local","scope":"local","status":"Success","code":0,"message":"Vlans
1111 created"}]}}
```

By default, the vRest APIs provide fabric level information. To specifically access the resources of the switch (scope : local), the switch ID needs to be specified in the URL. For switch ID specific information, use the command:

```
$ curl -u network-admin:pluribus_password http://switch1/vRest/vlans?
api.switch={hostid} | python -m json.tool
```

as in the following example:

```
$ curl -u network-admin:pluribus_password http://10.110.0.48/vRest/vlans?
api.switch=201327131 | python -m json.tool
```

For switch information listing a local scope:

```
$ curl -u network-admin:pluribus_password http://switch1/vRest/vlans?
api.switch=fabric | python -m json.tool
```

or

```
$ curl -u network-admin:pluribus_password  http://switch1/vRest/vlans |
python -m json.tool
```

as in the following example:

```
$ curl -u network-admin:pluribus_password  http://10.110.0.48/vRest/vlans |
python -m json.tool
```

> **Note:** Results returned may be an array.

About Pluribus Networks

Pluribus Networks delivers an open, controllerless software-defined network fabric for modern data centers, multi-site data centers, and distributed cloud edge environments.

The Linux-based Netvisor® ONE operating system and the Adaptive Cloud Fabric™ have been purpose-built to deliver radically simplified networking and comprehensive visibility along with white box economics by leveraging hardware from our partners Celestica, Dell EMC, and Edgecore, as well as Pluribus Networks' Freedom™ Series of switches.

The Adaptive Cloud Fabric provides a fully automated underlay and virtualized overlay with comprehensive visibility and brownfield interoperability and optimized to deliver rich and highly secure per-tenant services across data center sites with simple operations having no single point of failure.

Further simplifying network operations is Pluribus UNUM™, an agile, multi-functional web management portal that provides a rich graphical user interface to manage the Adaptive Cloud Fabric. UNUM has two key modules - UNUM Fabric Manager for provisioning and management of the fabric and UNUM Insight Analytics to quickly examine billions of flows traversing the fabric to ensure quality and performance.

Pluribus is deployed in more than 275 customers worldwide, including the 4G and 5G mobile cores of more than 75 Tier 1 service providers delivering mission-critical traffic across the data center for hundreds of millions of connected devices. Pluribus is networking, simplified.

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

Follow us on Twitter @pluribusnet or on LinkedIn at https://www.linkedin.com/company/pluribus-networks/

Corporate Headquarters

Pluribus Networks, Inc.
5201 Great America Parkway, Suite 422
Santa Clara, CA 95054

1-855-438-8638 / +1-650-289-4717

India Office

Pluribus Networks India Private Limited
Indiqube Brigade Square, 4th Floor
21, Cambridge Road
Bangalore 560008

Document Version - June 2020