GigaVUE-FM REST API User's Guide

GigaVUE-FM 3.5.00

COPYRIGHT

TRADEMARK ATTRIBUTIONS

DOCUMENT REVISION – 12/7/16

Contents

About This Guide

This guide is an introduction to the Application Program Interfaces (APIs) for the GigaVUE® Fabric Manager (GigaVUE-FM) and provides an overview of these REST APIs, basic work flows, and use cases. The APIs are implemented with the Representational State Transfer (REST) architecture.

Audience

This guide is intended for application developers interested in developing tools with the GigaVUE-FM APIs. Technical personnel familiar with a scripting or programming language can also use this guide. Familiarity with HTTP is a prerequisite to using the APIs.

Other Sources of Information

Gigamon provides additional documentation for the GigaVUE Fabric Manager (FM) on the Gigamon Customer Portal:

Document	Summary
GigaVUE-FM User's Guide	Describes how to install, deploy, and operate the GigaVUE-FM fabric management system.
GigaVUE-FM Release Notes	Describes new features and known issues in the release.
GigaVUE-FM API Reference	Describes the GigaVUE-FM REST APIs.
Visibility Platform for AWS Getting Started Guide	Describes how to deploy and configure the Visibility Platform for Amazon Web Services (AWS) with GigaVUE-FM.
GigaVUE-OS CLI User's Guide	Describes how to configure and operate the GigaVUE-OS software from the command-line interface.

Contacting Technical Support

Refer to http://www.gigamon.com/support-and-services/contact-support for Technical Support hours and contact information. You can also email Technical Support at support@gigamon.com.

Premium Support

Email Gigamon at inside.sales@gigamon.com for information on purchasing 24x7 Premium Support for your GigaVUE Traffic Visibility Node. Premium Support entitles you to round-the-clock phone support with a dedicated Support Engineer every day of the week.

Contacting Sales

Telephone: +1.408.831.4025

Sales: inside.sales@gigamon.com

1 Introduction to the GigaVUE-FM APIs

This chapter provides an introduction to the GigaVUE-FM APIs, which are designed with the Representational State Transfer (REST) architecture. Refer to the following sections:

- *Overview* on page 7
- *Installation* on page 8
- *What is REST?* on page 8
- *Authentication* on page 8
- *Resources* on page 9
- *GigaVUE-FM Resources* on page 12
- *HTTP* on page 15
- *GigaVUE-FM API Reference* on page 19

Overview

The GigaVUE-FM APIs provide a well structured architecture for performing query, update, and delete functions in a programmatic manner.

The APIs use the REST architecture, which makes it possible to create client applications that are platform- and language-independent. REST applications can use the APIs to perform various tasks with GigaVUE-FM by making requests and receiving responses as shown in Figure 1-1.

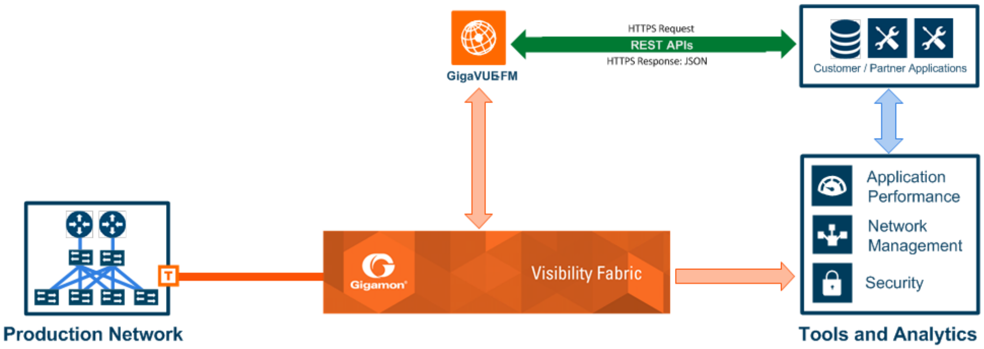

Figure 1-1: APIs and GigaVUE-FM

Applications that implement the GigaVUE-FM APIs can do the following:

- Improve security through better network detection, reaction, and response by automating NetFlow generation and SSL decryption so that current security appliances are not overtaxed when performing deep packet inspection. For example, security administrators can use the APIs that program the Visibility Fabric to dynamically change the traffic forwarding policies in response to threats or anomalous network traffic changes.
- For many organizations, IT Operations Management (ITOM) groups can use the APIs to develop programs that automate the following processes through software-defined visibility:
    - Performing common tasks, such as provisioning and ticketing of network port configurations.
    - Monitoring new IP subnets and VLANs.
    - Upgrading software images.

Installation

For support and installation instructions, refer to the *GigaVUE-FM and GigaVUE-VM User's Guide*.

What is REST?

REST is an architectural style that relies on a client-server communication protocol that is stateless. It is assumed that the client and server do not know anything about the state of an object other than what is communicated in requests and responses. Objects are identified by a uniform resource identifier (URI). Objects represent resources, and the URI provides a link to the resource. The state of a resource is communicated by structured documents passed between the client and the server.

GigaVUE-FM APIs use HTTP as the communication protocol to communicate with the GigaVUE Fabric Manager (FM). Therefore, anyone who is creating applications should be familiar with the HTTP protocol before creating clients that use the APIs.

Another important concept is Hypermedia as the Engine of State (HATEOS). HATEOS is a constraint on the REST architecture. This constraint states that hypertext is used to change the state of an application and allows the application to navigate through an API.

Authentication

Authentication must accompany each request that an application makes to GigaVUE-FM. The authentication is contained in the authorization header of the request. (For more information about headers, refer to *HTTP Headers* on page 15.) The APIs only support Basic authorization. Basic authorization uses unencrypted base64 encoded text to send the user name and password in the request.

The following is an example of using base64 encoding to encode a user name and password:

```
encoding = base64.base64encode('username1:password1')
print encoding
dXNlcm5hbWU6cGFzc3dvcmQ=
```

Notes:

- When an application logs in to GigaVUE-FM to make a request, the application should also log out after the request is complete. Otherwise, too many user sessions may be created and an invalid credential error message returned if the current request exceeds the maximum number of user sessions.
- Starting with GigaVUE-OS 4.7.00, the default admin password admin123A! is no longer allowed on nodes. When authenticating through the APIs against a node that is running GigaVUE-OS 4.7.00 or later, if admin123A! is used, the authentication will not complete.

Resources

Resources are a key component in REST and represent an object within a system. They are associated with data and have relationships to other resources. There is also a set of methods that operate on a resource.

The following sections provide an overview of these resources and the resources an application can access through GigaVUE-FM:

- *Links to Resources* on page 9
- *Representation of Resources* on page 10
- *JSON Representation* on page 10
- *HTTP Headers* on page 15
- *HTTP Methods* on page 15

Links to Resources

A uniform resource identifier (URI) provides the link to a resource. The URI consists of a base URI and a request URI. The base URI for GigaVUE-FM APIs is as follows, where `<fmip>` is the IP address for GigaVUE-FM and <api_version> is the current version of the API:

```
<fmip>/api/<api_version>
```

The request URI is the portion of the URI used to perform an HTTP request. For example:

```
/inventory/ports
```

A complete URI is:

```
<fmip>/api/v1.3/inventory/ports
```

NOTE: You can use the API GET `<fmip>/api/version` to get the current API version. This API returns the following content in JSON format:

```
{"apiVersion": "v1.3", "schemaVersion": "v1.3"}
```

Representation of Resources

GigaVUE-FM REST APIs use JavaScript object notation (JSON) to represent resources in a structured document. This representation is used in the request and response bodies of API calls. The default format is JSON and the *GigaVUE-FM API Reference* provides JSON models and model schema to describe response classes and request bodies. For information on how to access the reference, refer to *GigaVUE-FM API Reference* on page 19.

When a client makes a request to a resource, the request specifies that it is using JSON in the request and that the response should return data in JSON. The format is specified in the header of the request. For more information about headers, see *HTTP Headers* on page 15.

JSON Representation

JSON is a language-independent data format that is considered easy to read and write. The data is provided as a collection of unordered name/value pairs and ordered lists of values. The name/value pairs can be viewed as a dictionary and the ordered list of values as an array.

For example, the following represents port information in JSON format returned in a response body as the result of a query to the resource `/inventory/ports`:

```
{
   "context" : {
      "totalItems" : 59
   },
   "ports" : [ {
      "portId" : "5/2/x4",
      "comment" : "",
      "portType" : "tool",
      "adminStatus" : "down",
      "operStatus" : "down",
      "licensed" : true,
      "medium" : "OPTICAL",
      "configSpeed" : "10G",
      "duplex" : "full",
      "autoNeg" : false,
      "forceLinkUp" : false,
      "mtu" : 9600,
      "healthState" : "green",
      "neighborDiscovery" : "none"
   }, {
      "portId" : "5/2/x12",
      "comment" : "",
      "portType" : "network",
      "adminStatus" : "down",
      "operStatus" : "down",
      "licensed" : true,
      "medium" : "OPTICAL",
      "configSpeed" : "10G",
      "duplex" : "full",
      "autoNeg" : false,
      "forceLinkUp" : false,
      "mtu" : 9600,
      "healthState" : "green",
      "neighborDiscovery" : "none"
   }, ...
   ]
}
```

Because the data can be viewed as dictionaries and arrays, it is easy to extract the information from a response. For example, the following code processes the JSON document and prints the administrative and operational status of ports with an administrative status of *up*:

```
portInfo = json.loads(r.text)
ports = portInfo['ports']
for port in ports:
   adminStatus = port['adminStatus']
   operStatus = port ['operStatus']
   if adminStatus == 'up':
      print 'Port ID: ' + port['portId']
      print '  Admin Status: ' + port['adminStatus']
      print '  Op Status: ' +  port['operStatus']
```

In JSON, the collection of name/value pairs is referred to as an object, and the ordered collection of values is referred to as an array. In the previous code example, the `json.loads` function converts the JSON object into a dictionary.

Values are separated by commas and can be any of the following:

- A string in double quotes
- A number
- An object
- An array
- True
- False
- Null

GigaVUE-FM Resources

The GigaVUE-FM APIs provide much of the same functionality as the GigaVUE-OS CLI commands. Table 1-1 lists the available resources by their root and the CLI commands that provide a similar functionality as the APIs for a given resource. (For detailed information about the CLI commands, refer to the *GigaVUE-OS CLI User's Guide*. The base URI for the resources when making a request is `<fmip>/api/v3.1`. For more information about URIs, refer to *Links to Resources* on page 9.

Table 1-1: API Resources and CLI Commands

Resource	Description	CLI Command
/nodes	Node management	FM only
/nodeCredentials	Credentials for managed physical nodes	FM only
/inventory/chassis	Device chassis configuration	**show chassis**
/inventory/ports	Device port configuration	**show port**
/portConfig/portConfigs	Port configuration	**port**
/portConfig/portGroups	Port group configuration	**port-group**
/portConfig/gigastreams	GigaStream configuration	**gigastream**
/portConfig/portPairs	Port-pair configuration	**port-pair**
/portConfig/toolPortMirrors	Tool port mirror configuration	**tool-mirror**
/portConfig/toolPortFilters	Tool port filter configuration	**port filter rule**
/portConfig/stackLinks	Stack links configuration	**stack-link**
/portConfig/tunneledPorts	Tunneled port configuration	**tunneled-port**
/maps	Maps and map rules configuration	**map**
/mapChains	Map chain configuration	FM only
/mapTemplates	Map template configuration	**map-template**
/gsGroups	GigaSMART group configuration	**gsgroup**
/gsops	GigaSMART operations configuration	**gsop**
/vports	Virtual port configuration	**vport**
/apps/netflow	Netflow exporter, record, and monitor configuration	**apps netflow**
/apps/ssl	Secure Socket Layer (SSL) decryption keystore and decryption key configuration	**apps ssl**
/apps/gtp/whitelists	GTP Whitelist operations	**apps gtp-whitelist**
/apps/saApfProfiles	Session-aware APF profile configuration	**apps sapf**
/inline/networks	Inline network configuration	**inline-network**
/inline/networkGroups	Inline network group configuration	**inline-network-group**
/inline/tools	Inline tool configuration	**inline-tool**
/inline/toolGroups	Inline tool group configuration	**inline-tool-group**

Table 1-1: API Resources and CLI Commands

Resource	Description	CLI Command
/inline/serialToolGroups	Inline tool series configuration	**inline-serial**
/inline/hbProfiles	Heartbeat profile configuration	**hb-profile**
/inline/negativeHbProfiles	Negative heartbeat profile configuration	**inline-tool negative-heart-beat**
/inline/hbPackets	Heartbeat packet upload configuration	**hb-profile custom-packet** **hb-profile packet-format**
/inline/reduncancyProfiles	Inline tool group redundancy	**inline-network**
/system/config	Configuration files upload and download	**configuration fetch** **configuration upload**
/system/config/text	Text configuration files	**configuration text**
/system/sysdump	Sysdump file generation, download, and delete	**debug generate dump**
/system/time	Time settings	**ntp, ptp**
/system/syslogs	Syslog retrieval and update	**logging**
/system/localUsers	Local user account management	**username**
/system/security	System security policy settings	**system**
/system/aaa	Authorization and roles	**aaa authentication**
system/ldapServers	LDAP servers for authentication	**ldap**
/system/radiusServers	RADIUS servers for authentication	**radius-server**
/system/tacacsServers	TACACS+ servers for authentication	**tacacs-server**
/system/snmp	SNMP settings, v3 users, notification targets	**snmp-server host**
/system/email/notifications	Email notification settings	**email**
/system/uboot	Install binary bootloader code included with the active/booted image	**uboot install**
/clusterConfig	Physical cluster reboot and image upgrade	**reload**
/system/interfaces	Retrieve system interface information	**show interface**
/scheduledTasks	Retrieve scheduled tasks, schedule cluster configuration backups, and delete scheduled tasks.	FM only
/imageServers	Image file servers	FM only
/licensing	Licensing management for FM	FM only
/events	Manage events	FM only
/auditLog	Audit log of user actions on FM	FM only
/trending	Time series information for ports, maps, GS groups, virtual ports, GVM ports, and GVM maps	FM only

Table 1-1: API Resources and CLI Commands

Resource	Description	CLI Command
/trafficAnalyzer	Flow mapping analytics for dropped traffic	FM only
/tunnelEndpoints	Tunnel endpoint information	**tunneled-port**
/topology	Create, update, and delete manual nodes and links in Topology Visualization	FM only
/fmSystem	Reboot, backup, and restore GigaVUE-FM. Provide the image server and file path to the upgrade image on the server.	FM only
/fmSystem/archiveServers	Manage archive server and archive files.	FM only

Starting with GigaVUE-FM 3.5, Public Cloud Visibility APIs are available for use with the Gigamon Visibility Platform for AWS. These resources do not have any equivalent CLI commands. Table 1-2 lists the resources for Public Cloud Visibility. For details about these APIs, refer to the *GigaVUE-FM 3.5 API Reference*.

Table 1-2: Gigamon Visibility Platform for AWS API Resources

Resource	Description
vmm/ats/vms	Retrieve a list of VMs selected for a monitoring session.
vmm/configParams	Retrieve or modify configuration parameters.
vmm/tunnelSpecs	Retrieve, created, modify, or delete tunnel specification.
vmm/tunnelEndpoints	Retrieve tunnel endpoints.
vmm/maps	Retrieve, create, modify, or delete maps.
vmm/mapFolders	Retrieve, created, modify, or delete map folders and retrieve maps associated with a folder.
vmm/monitoringSessions	Retrieve, created, modify, or delete monitoring sessions.
vmm/stats	Retrieve monitoring session traffic statistics.
vmm/apps	Retrieve, register, update, or delete GigaSMART applications.
vmm/aws/connections	Retrieve, updated, add, or connect to AWS connections.
vmm/aws/inventory	Retrieve AWS inventory or rediscovery AWS inventory.
vmm/aws/gvTapControllers	Retrieve, add, modify, or deleted G-vTAP controller launch specifications and retrieve G-vTAP controllers.
vmm/aws/vseriesControllers	Retrieve, add, modify, or deleted V Series controller launch specifications and retrieve V Series controllers.
vmm/aws/vseriesNodes	Retrieve, add, modify, or deleted V Series node launch specifications and retrieve V Series nodes.

HTTP

The Hypertext Transfer Protocol (HTTP) is the protocol used in REST for communication between the client and the server. This section provides a description of the HTTP methods supported by the GigaVUE-FM APIs, HTTP headers, and status codes that can be returned in response to a request.

This section covers the following topics:

- *HTTP Headers* on page 15
- *HTTP Methods* on page 15
- *HTTP Status Codes* on page 19

HTTP Headers

HTTP headers are part of requests and responses and provide information about the client and the server. The headers contain authorization information and specify the format of the content in a request or response body.

The following are the basic headers:

- Authorization—Specifies the authentication used in the request. Basic authorization sends the user name and password as unencrypted base64 encoded text. Digest authorization sends the password to the server in a hashed form. (The GigaVUE-FM APIs use only Basic authorization.)
- Accept—Specifies the format of the data expected in the response to a request.
- Content-Type—Specifies the format of the data in a request or response body. This header is used in POST, PUT, and PATCH operations. It is not used with a GET operation.
- Content-Length—Specifies the number of bytes in the content body of a request.

NOTE: In the current release, the Content-Type and Accept headers should always specify `application/json`.

HTTP Methods

REST applications use HTTP methods to perform create, read, update, and delete operations on resources, otherwise known as CRUD operations. The GigaVUE-FM REST APIs support the following HTTP methods:

- *GET*
- *POST*
- *PUT*
- *PATCH*
- *DELETE*

These operations are idempotent. In the context of REST, this means that the same request always produces the same result on the server. However, the response from the server may be different because it may have changed state between requests.

GET

A `GET` operation requests a resource for a representation of that resource. `GET` requests only retrieve data.

The following example retrieves a port by port ID:

```
GET <fmip>/api/<api_version>/inventory/ports/5_2_x8
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Accept: application/json
```

The previous request returns the following response in JSON format:

```
{
   "port" : {
   "portId" : "5/2/x8",
      "alias" : "Map_Source",
      "comment" : "",
      "portType" : "network",
      "adminStatus" : "up",
      "operStatus" : "up",
      "licensed" : true,
      "medium" : "OPTICAL",
      "sfp" : {
         "sfpType" : "sfp+ sr",
         "sfpPower" : " -1.86 ",
         "vendorName" : "GIGAMON SFP-532",
         "vendorSn" : "APM0M2D        ",
         "vendorPn" : "GMON8571D3BCL-G"
      },
      "configSpeed" : "10G",
      "duplex" : "full",
      "autoNeg" : false,
      "forceLinkUp" : false,
      "mtu" : 9600,
   }
}
```

POST

A `POST` operation creates a new resource. The resource is created at the location specified in the request. For example, the following request adds a GigaSMART operation at `/api/<api_version>/gsops`:

```
POST <fmip>/api/<api_version>/gsops?clusterId=10.115.152.54
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
Content-Length : 129
```

The following is an example of a body used in the previous request:

```
{
   "alias" : "masking",
   "gsGroup" : "GS1",
   "gsApps" : {
      "masking" : {
         "protocol" : "tcp",
         "offset" : 4,
         "pattern" : "b" ,
         "length" : 13
      }
   }
}
```

PUT

A PUT operation modifies a resource with data provided in the request. The resource is created with the given data at the URI in the request if the resource does not already exist.

For example, the following request replaces an existing map definition defined by its alias.

```
PUT <fmip>/api/<api_version>/maps/AASlice_Level1
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept: application/json
Content-Length : 597
```

The following is an example of a body used in the previous request:

```
{
   "maps" : [ {
      "alias" : "AASlice_Level1",
      "clusterName" : "10.115.152.54",
      "type" : "firstLevel",
      "subType" : "byRule",
      "srcPorts" : [ "10/1/g10", "10/1/g11", "10/1/g12", "10/1/g13"],
      "dstPorts" : [ "vp" ],
      "order" : 1,
      "rules" : {
         "dropRules" : [ ],
         "passRules" : [ {
            "ruleId" : 1,
            "bidi" : true,
            "matches" : [ {
               "type" : "portSrc",
               "value" : 80
         } ]
      }, {
         "ruleId" : 2,
         "bidi" : true,
         "matches" : [ {
            "type" : "portSrc",
            "value" : 20
         } ]
      }, {
         "ruleId" : 3,
         "bidi" : true,
         "matches" : [ {
            "type" : "portSrc",
            "value" : 21
         } ]
      } ]
   },
   "roles" : {
      "owners" : [ "admin" ],
         "viewers" : [ ],
         "editors" : [ ],
         "listeners" : [ ]
      }
   } ]
}
```

PATCH

A PATCH operation modifies individual properties of a resource according to the instructions in the request.

For example, the following request changes a tool port to a network port and enables the administrative status:

```
PATCH <fmip>/api/<api_version>/inventory/ports/3_1_x1
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type: application/json
Accept: application/json
Content-Length : 62
```

The following is an example of the body used in the previous request:

```
{
   'adminStatus': 'up',
   'portType': 'network',
   'portId': '3/1/x1'
}
```

DELETE

A DELETE operation removes a resource.

For example, the following request deletes a map by its alias:

```
DELETE <fmip>/api/<api_version>/maps/AASlice_Level1
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type: application/json
Accept: application/json
```

HTTP Status Codes

A response to a request provides a status code indicating to the client whether the request was successful or failed. Table 1-3 lists some of the status codes that the GigaVUE-FM APIs can return.

Table 1-3: HTTP Status Codes

Status Code	Meaning
200 OK	The request succeeded.
201 Created	The request was accepted and a new resource was created.
202 Accepted	The server accepted the request.
204 No Content	The response did not have any content.
400 Bad Request	The request is invalid.
401 Unauthorized	The request was not authenticated.
403 Forbidden	Access denied.
404 Not Found	The entity was not found.
409 Conflict	The entity already exists.
500 Internal Server	An error occurred on the server.

For 400 and 500 series status codes, the response from GigaVUE-FM contains a body in JSON format that provides a description of the error.

GigaVUE-FM API Reference

You can access the descriptions of the GigVUE-FM Core APIs from GigaVUE-FM by clicking on the appropriate link under Support. The help page lists the APIs for the different types of operations. Each description provides the methods available, the JSON model and model schema for the requests and responses, and the applicable HTTP status codes returned in error messages. You can also connect to the on-line *GigaVUE-FM API Reference* through the following URL:

```
<FM IP address>/apiref/apiref.html
```

Response Error Messages

Starting with GigaVUE-FM version 3.4, REST API requests return a global error message if an error occurs. Prior to GigaVUE-FM version 3.4, if an error occurred the error message presented was the error returned by the device, which could vary from device to device for the same error condition. The global error message returned by GigaVUE-FM APIs have an error response body that contains a unique code and a message. The following is an example:

```
"errors": [{
    "code":"0x800200b",
    "msg": “Invalid alias 'Test Space'. ''is a reserved character."
}]
```

The following tables list the possible error codes and their descriptions.

Table 1-4: Gigamon Device Error Codes

Error Code	Description
0x80020000	initialization failure
0x80020001	TMS internal failure
0x80020002	Null Objects
0x80020003	No more resource
0x80020004	uninitialized param
0x80020005	Unsupported Product Code
0x80020006	Unsupported Operation
0x80020007	Configuration Failed
0x80020008	Configuration exists
0x80020009	Configuration not found
0x8002000a	Operation not allowed on reserved object
0x8002000b	invalid argument
0x8002000c	invalid net addr/mask
0x8002000d	invalid ip addr/mask
0x8002000e	GV DB query failed
0x8002000f	GVO internal failure
0x80020010	Objects exceeds maximum range
0x80020011	Config in progress
0x80020012	User invalid
0x80020013	User role Invalid
0x80020014	User Permission Insuffcent for Operation
0x80020015	Unlicensed Feature

Table 1-4: Gigamon Device Error Codes

Error Code	Description
0x80020016	Feature License expired
0x80020017	Chassis is in SAFE mode! No new traffic configuration is allowed.
0x80020018	Chassis is in LIMITED mode! The configuration is not allowed.
0x80020020	Chassis not valid
0x80020021	Card not valid
0x80020022	Card Mode Invalid
0x80020023	Slot not valid
0x80020024	Slot State Invalid
0x80020025	Slot not configured
0x80020026	Unconfigured box-id
0x80020027	Chassis mode is not supported
0x80020028	CC card does not support this card
0x80020030	Port Invalid
0x80020031	Port Type Invalid
0x80020032	Port not configured
0x80020033	Port Mode Invalid
0x80020034	Port Inactive
0x80020035	Port SFP Unsupported
0x80020036	Port Config is invalid
0x80020037	Port Config exists
0x80020038	Port conflict in config
0x80020039	Portpair update disallowed. Config exists
0x8002003a	Portpair object mismatch
0x8002003b	Portpair cross-box not supported
0x8002003c	Portpair cross-card not supported
0x8002003d	Portpair with tunnel not supported
0x8002003e	Portpair Linktimer failed
0x8002003f	Port Group in Use
0x80020040	Port Group Object Mismatch
0x80020041	Port Group has Tunnel config
0x80020042	Port Group resource exceeds range
0x80020043	Stacklink in Use
0x80020044	Stacklinks cannot be in same box

Table 1-4: Gigamon Device Error Codes

Error Code	Description
0x80020045	Stacklinks not found
0x80020046	Stacklinks will cause loops
0x80020047	Port in Use
0x80020048	Invalid port to break out
0x80020060	GSOP required for operation
0x80020061	GSOP SSL requires order of operation
0x80020062	GSOP invalid
0x80020063	GSOP invalid GSAPP combination
0x80020064	GSOP Configuration exists
0x80020065	GSOP object mismatch
0x80020066	GSOP AFP needs VPORT
0x80020067	GSOP and VPORT not in same group
0x80020068	GSOP and GSGroup mismatch
0x80020069	GSOP needs flow filter
0x80020070	GS:internal error
0x80020071	GS:gsrule object mismatch
0x80020072	GS:gsrule and flowrule mismatch
0x80020073	GS: no tunnel-port
0x80020074	GS: Tunnel Config exits
0x80020075	GS: Tunnel object mismatch
0x80020076	GS: Tunnel feature not supported
0x80020077	GS: Tunnel resource not available
0x80020078	GS: Tunnel incomplete config
0x80020079	GS:SSL Error
0x8002007a	GS:SSL app not found
0x8002007b	GSGROUP SSL passwd not set
0x8002007c	GS:SSL key error
0x8002007d	GS:SSL Max key object
0x8002007e	GS:SSL duplicate key object
0x8002007f	GS:SSL failed to decrypt key object
0x80020080	GS:SSL failed to decrypt key object
0x80020081	GS:SSL Server config exists
0x80020090	GSGROUP not valid

Table 1-4: Gigamon Device Error Codes

Error Code	Description
0x80020091	GSGROUP Filter has more e-ports
0x80020092	GSGROUP Max num or GSRULES exceeds
0x80020093	GSGROUP Config exits
0x80020094	GSGROUP object not found
0x80020095	GSGROUP object mismatch
0x80020096	GSGROUP Flow-sampling has more e-ports
0x80020097	GSGROUP Ports are not in same chassis
0x80020098	GSGROUP Netflow object mismatch
0x80020099	GSGROUP Netflow object not found
0x8002009a	GSGROUP Netflow feature not support on V9
0x80020100	GigaStream config exists
0x80020101	GigaStream config in use
0x80020102	GigaStream config object mismatch
0x80020103	GigaStream not on same resource or entity
0x80020104	GigaStream resource exceeds range
0x80020180	Invalid rule
0x80020181	MAP object mismatch
0x80020182	MAP Max num or RULES exceeds
0x80020183	MAP: Passall with map objects not allowed
0x80020184	MAP rules with collector not allowed
0x80020185	MAP invalid gsrules
0x80020186	MAP config exists
0x80020187	MAP config invalid
0x80020188	MAP: VPORT object mismatch
0x80020189	MAP: VPORT config exists
0x8002018a	MAP inline port-list invalid
0x8002018b	MAP inline port config exists
0x8002018c	MAP inline source in port-list
0x8002018d	MAP: gsop in inline not allowed
0x8002018e	MAP:app in inline not allowed
0x8002018f	MAP: VPORT in inline not allowed
0x80020190	MAP: inline destination conflict
0x80020191	MAP: inline source conflict

Table 1-4: Gigamon Device Error Codes

Error Code	Description
0x80020192	MAP invalid gsrules
0x80020193	GSOP apps require reboot to take effect
0x80020194	GSOP Duplicate SSL configuration
0x80020195	GS: Tunnel Exporter Config exits
0x80020196	GS: Exporter Alias not found
0x80030000	initialization failure
0x80030001	invalid argument
0x80030002	uninitialized param
0x80030003	Unsupported Product Code
0x80030004	incomplete port string
0x80030005	port not found
0x80030006	Unknown error
0x80050000	initialization failure
0x80050001	invalid argument
0x80050002	uninitialized param
0x80050003	Unsupported Product Code
0x80050004	Unknown error
0x80070000	initialization failure
0x80070001	invalid argument
0x80070002	uninitialized param
0x80070003	Unsupported Product Code
0x80070004	Unknown error
0x80090000	initialization failure
0x80090001	invalid argument
0x80090002	uninitialized param
0x80090003	Unsupported Product Code
0x80090004	Unknown error
0x800a0000	initialization failure
0x800a0001	invalid argument
0x800a0002	uninitialized param
0x800a0003	invalid param
0x800a0004	User provided buffer is NULL
0x800a0005	Stats Buffer Overflow(less that require buffer size)

Table 1-4: Gigamon Device Error Codes

Error Code	Description
0x800a0006	Port number out of Range
0x800a0007	Out of Range (Parameter)
0x800a0008	Unknown Stat type
0x800a0009	Provided History count does not match Stat type
0x800a000a	Stat Query: for Device failed
0x800a000b	Stat Query: send request timed out
0x800a000c	Stat Query: recv request timed out
0x800a000d	Stat Query: remote node query failed
0x800a000e	Stat Query: remote node returned nothing
0x800b0000	initialization failure
0x800b0001	invalid argument
0x800b0002	uninitialized param
0x800b0003	Unsupported Product Code
0x800b0004	Unknown error
0x800c0000	initialization failure
0x800c0001	invalid argument
0x800c0002	uninitialized param
0x800c0003	invalid param
0x800c0004	incomplete port string
0x800c0005	invalid port type
0x800c0006	mod_id is out of range
0x800c0007	mod_id un-assigned
0x800c0008	box-id out of range
0x800c0009	slot-id out of range
0x800c000a	card-index out of range
0x800c000b	port not found
0x800c000c	Unknown error
0x800d0000	command terminated - not shown to the user.
0x800d0001	Unknown error
0x800e0000	ugw invalid request
0x800e0001	ugw session not authentication
0x800e0002	ugw access denied
0x800e0003	ugw entity no found

Table 1-4: Gigamon Device Error Codes

Error Code	Description
0x800e0004	ugw entity already exists
0x800e0005	ugw invalid cookie
0x800e0006	mgmtd internal error
0x800e0007	ugw not implemented
0x800e0008	Unknown error

Table 1-5: REST API Errors

Error Code	Description
0x800f0000	general: The requested URL was not found on this server
0x800f0001	general: Login failure
0x800f0002	general: Session ID not valid. No need to logout.
0x800f0003	general: Failed to connect to the unified gateway.
0x800f0004	general: The Content-Type is application/json but the request body is not valid JSON.
0x800f0005	general: Only application/json Content-Type is supported for this operation.
0x800f0006	general: JSON schema validation failed on request body
0x800f0007	general: No request body present. This operation requires a JSON request body.
0x800f0008	general: The request body is not valid JSON object. This operation requires a JSON object request body.
0x800f0009	general: Not implemented
0x800f000a	general: Unauthorized
0x800f000b	general: Forbidden
0x800f000c	general: Resource not found
0x800f000d	general: Incorrect alias
0x800f000e	general: Invalid alias
0x800f000f	general: Resource exists
0x800f0010	general: Invalid action
0x800f0011	general: Missing required param
0x800f0012	general: The page query parameter must be in the form '(pageNum:pageSize)'.
0x800f0013	general: Invalid sort query
0x800f0014	general: Writing to a readonly property

Table 1-5: REST API Errors

Error Code	Description
0x800f0015	general: Cli error
0x800f0016	apps/netflow/exporter/filter: Bid/sid of valueMax must equal bid/sid of value
0x800f0017	apps/netflow/exporter/filter: Input port error
0x800f0018	apps/netflow/exporter/filter: Ipv6 requires a value for one of 'netMask' or 'valueMax'.
0x800f0019	system/snmp: Incorrect notif target address
0x800f001a	system/snmp: Incorrect username
0x800f001b	system/config/text: Username and password is required
0x800f001c	portConfig/portConfigs: Invalid port threshold
0x800f001d	portConfig/portConfigs: Invalid tool port
0x800f001e	mapChains: Incorrect map chain id
0x800f001f	inventory/chassis: Incorrect slotid
0x800f0020	inventory/chassis: Invalid slotid
0x800f0021	apps/netflow/records: Record invalid
0x800f0022	portConfig/portGroups: If portWeights is included, the list size must match the size of the 'ports' list.
0x800f0023	apps/ssl/keystore: Keystore password is already set
0x800f0024	apps/ssl/keystore: Keystore password is not set
0x800f0025	apps/ssl/keyMaps: Alias in the SslDecryptionKeyMapConfigSpec body must reference one of the existing GsGroups
0x800f0026	gsGroups: Invalid keymap
0x800f0027	general: 'time' must be in ISO-8601 date format 'yyyy-MM-dd'T'HH:mm:ssZ'
0x800f0028	maps: Invalid srcport
0x800f0029	maps: Invalid dstport
0x800f002a	maps: Incorrect ruleid
0x800f002b	maps: Only one of rules, gsRules, flowRules may be set.
0x800f002c	maps: Invalid ipv4
0x800f002d	maps: Invalid ipv6
0x800f002e	maps: Invalid cidrmask
0x800f002f	maps: Invalid mac
0x800f0030	maps: RuleMatching is only valid for regular/byRule map types
0x800f0031	maps: Asymmetric inlineTrafficType is only applicable for inline/passAll maps
0x800f0032	maps: InlineTrafficPath is only applicable for inline maps

Table 1-5: REST API Errors

Error Code	Description
0x800f0033	maps: TrafficType is only applicable to firstLevel map types
0x800f0034	maps/rules: Included rule elementes are mutually exclusive to each other
0x800f0035	maps/rules: 'valueMax' must be greater than 'value'
0x800f0036	maps/rules: Rule element is only allowed for gsRule.
0x800f0037	maps/rules: 'subset' is only valid when 'valueMax' is present.
0x800f0038	maps/rules: Value for pos is repeated
0x800f0039	gsops: TunnelDecap of type gmip requires value for gmipPort
0x800f003a	inline/networks: Invalid delete
0x800f003b	inline/serialToolGroups: Invalid inline tool
0x800f003c	inline/networkGroups: Invalid inline network
0x800f003d	gsops: FabricPath requires both fpSrcSwitchId and fpDstSwitchId
0x800f003e	gsops: Invalid gsapp
0x800f003f	gsops: Fm6000Ts requires timestampFormat
0x800f0040	system/aaa/auth: UnlockTime must be greater than or equal to lockTime
0x800f0041	apps/netflow/monitor: Invalid sampling space
0x800f0042	apps/saApfProfiles: Session field pos unhandled
0x800f0043	apps/saApfProfiles: Session field type unhandled
0x800f0044	apps/saApfProfiles: IPv4, and IPv6 field cannot be mixed in a session field
0x800f0045	inline/hbPackets: Size
0x800f0046	portConfig/portConfigs: Invalid ingress port vlan tag: 1
0x800f0047	portConfig/gigastreams: HC2 allows advanced-hash configuration for slot 'cc1' only. And it is shared across line-card modules.
0x800f0048	gsops: LbType 'gtpKeyHash' is only applicable when 'appType' is 'gtp'
0x800f0049	system/localUsers: Rsvd username
0x800f004a	apps/gtp/whitelists: System is busy processing GTP whitelist. Try again later.
0x800f004b	apps/gtp/whitelists: Exceed entry limit
0x800f004c	apps/gtp/whitelists: Exceed per file limit
0x800f004d	apps/gtp/whitelists: Cannot import file with invalid GTP whitelist entries or incorrect format
0x800f004e	apps/gtp/whitelists: Destroy in use

Table 1-5: REST API Errors

Error Code	Description
0x800f004f	apps/gtp/whitelists: The value for 'usage' must be one of 'create', 'delete'
0x800f0050	system/aaa/roles: A builtin role cannot be deleted or created.
0x800f0051	system/snmp: Invalid ipv4 ipv6
0x800f0052	apps/saApfProfiles: Invalid value for 'bufferSize': 1
0x800f0053	system/config: Directory not found
0x800f0054	system/snmp/notifTargets: V3User is required when version is 'v3'
0x800f0055	system/snmp: Only one communityString allowed when enableMultiCommunity is False
0x800f0056	gsops: Invalid offset
0x800f0057	gsops: Invalid combo
0x800f0058	gsops: FlowFilter, flowSample gtp, or gtpWhitelist must be configured for loadBalance stateful appType 'gtp'
0x800f0059	gsops: One and only one of stateful or stateless must be configured for loadBalance
0x800f005a	gsops: GsApp 'saApf' requires 'apf' to be enabled
0x800f005b	gsops: GsApp 'headerRemove' protocol 'isl' is not compatible with 'dedup' or 'tunnelDecap'
0x800f005c	system/license: Install error
0x800f005d	stats/port: Stat error
0x800f005e	apps/saApfProfiles: Redefine fail
0x800f005f	apps/netflow/exporters: Exporter in use
0x800f0060	inventory/chassis: Card mode fail
0x800f0061	gsGroups: Invalid watchdog timer
0x800f0062	general: Missing argument
0x800f0063	general: Unexpected exception
0x800f0064	general: Exception occurred in write_error(). Check the logs for more information.
0x800f0065	general: Method Not Allowed.
0x800f0066	general: Http error
0x800f0067	gsops: Erspan requires erspanFlowId
0x800f0068	/inline/negativeHbProfiles: Empty custom packet
0x800f0069	/inline/negativeHbProfiles: Invalid base64
0x800f006a	general: Admin account password must be changed via the CLI to a non-default value for security purposes.

Table 1-5: REST API Errors

Error Code	Description
0x800f006b	general: Bad filename
0x800f006c	Unknown error

Table 1-6: General Error Codes

Error Code	Description
0x80100000	No error
0x801036b0	Generic error
0x801036b1	Unexpected null
0x801036b2	Assertion failed
0x801036b3	Nyi
0x801036b4	Not found
0x801036b5	Exists
0x801036b6	Multiple
0x801036b7	Unexpected eof
0x801036b8	Unexpected arg
0x801036b9	Bad path
0x801036ba	Bad file
0x801036bb	No buffer space
0x801036bc	Parse error
0x801036bd	File not found
0x801036be	Io error
0x801036bf	Io blocking error
0x801036c0	Io closed
0x801036c1	Io timeout
0x801036c2	Io bad existence
0x801036c3	Io verify failed
0x801036c4	Io bad family
0x801036c5	Unexpected case
0x801036c6	Bad type
0x801036c7	Bad utf8 seq
0x801036c8	Divide by zero
0x801036c9	No fs space

Table 1-6: General Error Codes

Error Code	Description
0x801036ca	Decryption failure
0x801036cb	Read only
0x801036cc	Java exception
0x801036cd	Limit exceeded
0x801036ce	Too many fds
0x801036cf	Not initialized
0x801036d0	Lock busy
0x801036d1	Lock not locked
0x801036d2	Lock bad owner
0x801036d3	Range
0x801036d4	Invalid param
0x801036d5	Skip value
0x8010370a	Has children
0x8010370b	Not an ancestor
0x8010370c	No path
0x80103714	Foreach delete
0x80103715	Foreach remove
0x80103716	Foreach halt err
0x80103717	Foreach halt ok
0x8010371e	Reset timeout
0x8010371f	Event delete
0x80103723	Not cli shell
0x80103724	Cli cmd too long
0x80103728	Cancelled
0x80103729	Cancelled error
0x8010372d	Wrong num instances
0x80103779	Transient failure
0x801037dd	Db unrecognized type
0x801037de	Bw not available
0x801037df	Unknown error

2 GigaVUE-FM API Workflows

This chapter describes the basic workflow for a REST application and GigaVUE-FM using the APIs. The workflows described are:

- *REST Workflow* on page 33
- *GigaVUE-FM Workflow* on page 34

REST Workflow

Applications using the REST architecture follow a workflow that has two operations, which are as follows:

1. Make a request to the server to create, read, update, or delete a resource.
2. Handle the response to the request.

   The response contains a status code, indicating whether the request succeeded or failed. The response can also contain data in a structured format, in addition to the status code. The application either takes further actions based on the status code or processes the data.

The application repeats the cycle of request and response as long as necessary, as shown in Figure 2-1.

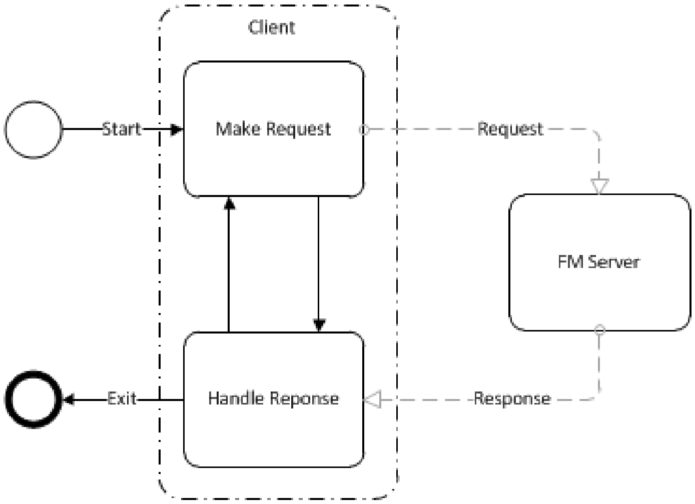

Figure 2-1: REST Workflow

GigaVUE-FM Workflow

This section describes how to use the GigaVUE-FM APIs in a general workflow. The topics covered are as follows:

- *Managing Physical Nodes* on page 34
- *Configuring Ports* on page 35
- *Configuring Traffic Maps* on page 39

Logging In and Logging Out

When using the GigaVUE-FM APIs, a script or application needs to first log in to GigaVUE-FM. This can be done creating a session for making a series of requests or logging prior to each individual request. In each case, the script or program *must end the session or log out when it is finished making a request or requests* to GigaVUE-FM. Failure to log out from GigaVUE-FM can result in an invalid credential error message being returned if the current request exceeds the maximum number of user sessions.

Managing Physical Nodes

The `/nodes` resource represents a physical node. Nodes can belong to a cluster. With the Gigamon node APIs, an application can perform the following operations on nodes:

- Add nodes to GigaVUE-FM and rediscover them. (Refer to *Adding Nodes* on page 34.)
- Retrieve information about nodes from GigaVUE-FM. (Refer to *Getting Node Information* on page 35.)
- Remove node clusters from GigaVUE-FM. (Refer to *Deleting Clusters or Nodes* on page 35.)

Adding Nodes

An application can add nodes to GigaVUE-FM. The node is added through its node address. If the node is part of a cluster, it joins the cluster. If the node address is for a standalone node, that node is added to GigaVUE-FM.

The `/nodes` resource only provides information about the nodes. It does not provide any information about physical characteristics of the nodes. To get the physical characteristics, use a `GET` operation on the `/inventory/chassis` or `/inventory/ports` resource or both.

To add a node, an application does the following:

1. Makes a `POST` request to the node resource.

   ```
   POST <fmip>/api/<api_version>/nodes
   ```

   The request needs to contain a payload that specifies a node address. For example, the following request body specifies that a node with address 10.115.152.54 is added:

   ```
   {
   ```

```
   "nodeAddSpecs”: [{
      "nodeAddress" : "10.115.152.54",
      "username" : "admin",
      "password" : "admin123A!"
   ]}
}
```

If a failure occurs with adding the node, it is indicated in the response body. (For a description of the error message, refer to the *GigaVUE-FM API Reference*.)

2. Repeats step 1 as necessary to add additional nodes.
3. Rediscovers the nodes with the following `PUT` request:

   ```
   PUT <fmip>/api/<api_version>/nodes
   ```

   NOTE: This API does not include a content body in the request.

Getting Node Information

An application can use the GigaVUE-FM APIs to list the information about clusters and nodes. The APIs can return the information as a list of node clusters or as a flat list of nodes.

To retrieve a list of nodes clusters, an application uses the following request:

```
GET <fmip>/api/<api_version>/nodes
```

To retrieve a list of the nodes without grouping them by cluster, an application uses the following request:

```
GET <fmip>/api/<api_version>/nodes/flat
```

To retrieve information about a specific cluster, include the cluster ID in the request:

```
GET <fmip>/api/<api_version>/nodes?clusterId=<clusterId>
```

To retrieve information about a specific node, request a flat list of nodes and specify the node ID:

```
GET <fmip>/api/<api_version>/nodes/flat?nodeId=<nodeId>
```

Deleting Clusters or Nodes

To delete all clusters and standalone nodes, an application uses the following request:

```
DELETE <fmip>/api/<api_version>/nodes
```

To remove an entire cluster, an application uses the following request:

```
DELETE <fmip>/api/<api_version>/nodes?clusterId=<clusterId>
```

If `clusterId` identifies a standalone node, only the standalone node is deleted.

Configuring Ports

After an application adds nodes to GigaVUE-FM, the next step is to configure the ports. An application can use the APIs to create, show, update, and delete ports. In addition to adding port groups, an application can configure GigaStream. A GigaStream groups together multiple tool ports into logical bundles, allowing traffic to be distributed to multiple monitoring tools.

This following sections describes the tasks that an application can perform with the APIs on ports and GigaStream:

- *Getting Port Information* on page 36
- *Setting the Port Type and Enabling* on page 37
- *Creating Port Groups* on page 37
- *Getting Port Group Information* on page 37
- *Updating Port Groups* on page 37
- *Deleting a Port Group* on page 38
- *Creating GigaStream* on page 38
- *Getting GigaStream Information* on page 38
- *Deleting GigaStream* on page 38

Getting Port Information

To retrieve information about all ports, an application uses the following request:

```
GET <fmip>/api/<api_version>/inventory/ports?clusterId=<clusterId>
```

To retrieve information about a specific port, an application uses the following request:

```
GET <fmip>/api/<api_version>/inventory/ports/
{portId}?clusterId=<clusterId>
```

The following is an example of the response returned from a request for information about a specific port, where the port ID is 5/2/x1:

```
{
   "port" : {
      "portId" : "5/2/x1",
      "alias" : "Spirent_P17",
      "comment" : "",
      "portType" : "network",
      "adminStatus" : "up",
      "operStatus" : "up",
      "licensed" : true,
      "medium" : "OPTICAL",
      "sfp" : {
         "sfpType" : "sfp+ sr",
         "sfpPower" : " -2.83 ",
         "vendorName" : "GIGAMON SFP-532",
         "vendorSn" : "MSP0GWZ        ",
         "vendorPn" : "GMON8571D3BCL-G"
      },
      "configSpeed" : "10G",
      "operSpeed" : "10G",
      "duplex" : "full",
      "autoNeg" : false,
      "forceLinkUp" : false,
      "mtu" : 9600,
        "healthState" : "green",
        "neighborDiscovery" : "none"
    }
}
```

Setting the Port Type and Enabling

To set the port type and enable it, an application uses the following request:

```
   PATCH <fmip>/api/<api_version>/inventory/ports/{
portId}?clusterId=<clusterId>
```

The payload of the request specifies the port by its ID, the port type, and its administrative status. For example, the following payload makes port 5/2/x8 a network port with its administrative status enabled.

```
   {
      "portId" :  "5/2/x8",
      "portType" : "network",
      "adminStatus" : "up"
   }
```

Creating Port Groups

To create a port group, an application makes the following request:

```
   POST <fmip>/api/<api_version>/portConfig/
portGroups?clusterId=<clusterId>
```

This request needs to include a payload that describes the port group. For example, the following request body creates a port group with two ports:

```
{
   "alias" : "PG0",
   "ports" : [ "5/4/x4", "5/4/x1" ],
   "comment": " ",
}
```

The payload specifies that ports 5/4/x4 and 5/4/x1 are in the port group and the alias of the group is PG0.

Getting Port Group Information

To retrieve a list of the port groups, an application makes either of the following requests:

- `GET <fmip>/api/<api_version>/portConfig/portGroups?clusterId=<clusterId>`
- `GET <fmip>/api/<api_version>/portConfig/portGroups/{alias}?clusterId=<clusterId>`

Updating Port Groups

An application can update a port group through a PUT or a PATCH operation. The PUT operation replaces the current configuration of the port group. A PATCH operation modifies the port group. In general, a PUT operation is preferred over a PATCH.

To update or modify a port group, an application can use either of the following requests:

- `PUT <fmip>/api/<api_version>/portConfig/portGroups/{alias}?clusterId=<clusterId>`
- `PATCH <fmip>/api/<api_version>/portConfig/portGroups/{alias}?clusterId=<clusterId>`

This request needs to include a payload that describes the port group. For example, the following request body changes one port in the port group (5/4/x5 replaces 5/4/x1) and adds a third port to port group PG0:

```
{
   "alias" : "PG0",
   "ports" : [ "5/4/x4", "5/4/x5", "5/4/x6" ],
   "comment": " ",
}
```

NOTE: A `PUT` operation modifies the map with data provided in the request. A `PATCH` operation modifies individual properties of the map according to the instructions in the request. A `PUT` operation is preferred over a `PATCH` operation.

Deleting a Port Group

To a delete a port group, an application uses the following request:

```
   DELETE /portConfig/portGroups/{alias}?clusterId=<clusterId>
```

Creating GigaStream

To create a GigaStream, an application uses the following request:

```
   POST <fmip>/api/<api_version>/portConfig/
gigastreams?clusterId=<clusterId>
```

Getting GigaStream Information

An application can retrieve all GigaStream in a cluster or retrieve a specific GigaStream by its alias.

To retrieve all GigaStream, an application uses the following request:

```
   GET <fmip>/api/<api_version>/portConfig/
gigastreams?clusterId=<clusterId>
```

To retrieve a specific GigaStream, an application specifies the alias of the GigaStream in the request:

```
   GET <fmip>/api/<api_version>/portConfig/gigastreams/
   {alias}?clusterId=<clusterId>
```

Deleting GigaStream

To delete GigaStream, an application uses the `DELETE` operation and specifies the alias of the GigaStream in the following request:

```
   DELETE <fmip>/api/<api_version>/portConfig/gigastreams
      /{alias}?clusterId=<clusterId>
```

Configuring Traffic Maps

An application can use the GigaVUE-FM APIs to configure and modify traffic maps, which includes adding, defining, and deleting rules. Maps also use GigaSMART operations (gsop) that are assigned to a GigaSMART group (gsgroup).

Working with Maps

This section describes the following:

- *Creating Maps* on page 39
- *Getting Map Information* on page 40
- *Updating Maps* on page 40
- *Map Chains* on page 41

Creating Maps

To create a traffic map, an application uses the following request:

```
POST <fmip>/api/<api_version>/maps?clusterId=<clusterId>
```

The request contains a payload that describes the map. The following is an example of a payload that creates a map for passing IPv4 and IPv6 traffic:

```
{
   "alias" : "Map_RSA_Traffic",
   "type" : "regular",
   "subType" : "byRule",
   "srcPorts" : [ "5/2/x8" ],
   "dstPorts" : [ "5/2/x16" ],
   "order" : 1,
   "rules" : {
      "dropRules" : [ ],
      "passRules" : [ {
         "ruleId" : 1,
         "comment" : " ",
         "matches" : [ {
            "type" : "ipVer",
            "value" : "v4"
         } ]
      }, {
         "ruleId" : 2,
         "comment" : " ",
         "matches" : [ {
            "type" : "ipVer",
            "value" : "v6"
         } ]
      } ]
   },
   "ruleMatching" : "normal"
}
```

Getting Map Information

To retrieve a list of the maps, an application uses either of the following requests:

- `GET <fmip>/api/<api_version>/maps?clusterId=<clusterId>`
- `GET <fmip>/api/<api_version>/maps/{alias}?clusterId=<clusterId>`

The following is an example of the response returned from a request for information about a specific map, where the map alias is L1_Map_Mask:

```
{
   "map" : {
   "alias" : "L1_Map_Mask",
   "clusterId" : "10.115.152.50",
   "type" : "firstLevel",
   "subType" : "byRule",
   "srcPorts" : [ "5/2/x8" ],
   "dstPorts" : [ "vp5-1" ],
   "order" : 1,
   "rules" : {
      "passRules" : [ {
         "ruleId" : 1,
            "comment" : "",
            "bidi" : false,
            "matches" : [ {
               "type" : "macSrc",
               "value" : "0000.0000.0000",
               "mask" : "0000.0000.0000"
            } ]
         } ]
      },
      "roles" : {
         "owners" : [ "admin" ],
         "viewers" : [ ],
         "editors" : [ ],
         "listeners" : [ ]
      }
   }
}
```

Updating Maps

To make changes to an existing map, an application uses the following request:

```
   PUT <fmip>/api/<api_version>/maps/{alias}?clusterId=<clusterId>
```

The request contains a payload with the changes to the map.

The following example request and payload adds SSL decryption to the map shown in the previous section *Creating Maps*, where the alias specified in the request is Map_RSA_Traffic:

```
   payload = {
      "alias" : "Map_RSA_Traffic",
      "type" : "regular",
      "subType" : "byRule",
      "srcPorts" : [ "5/2/x8" ],
      "dstPorts" : [ "5/2/x16" ],
      "gsop" : "GSOP_SSL_RSA",
      "order" : 1,
      "rules" : {
         "dropRules" : [ ],
         "passRules" : [ {
            "ruleId" : 1,
            "comment" : "",
            "matches" : [ {
               "type" : "ipVer",
               "value" : "v4"
            } ]
         } ]
      },
      "ruleMatching" : "normal"
   }
```

```
PUT <fmip>/api/<api_version>/maps/
Map_RSA_Traffic?clusterId=<clusterId>
```

An application can also use a PATCH operation to make changes to a map. However, PUT operations are preferred for making changes to a resource. For a description of these operations, refer to *HTTP Methods* on page 15.

Map Chains

Map chains are traffic maps assigned to the same group of ports. All maps in a chain must share the same source ports, but they can have different destination ports. The maps in a map chain also have a priority. A chain can have a map without any rules that can be used as a shared collector.

Figure 2-2 shows a map chain with three maps assigned to a port group. The maps use the same source ports and different destination ports. The red X indicates a connection to a source port not allowed in the chain because it is not shared by the other maps.

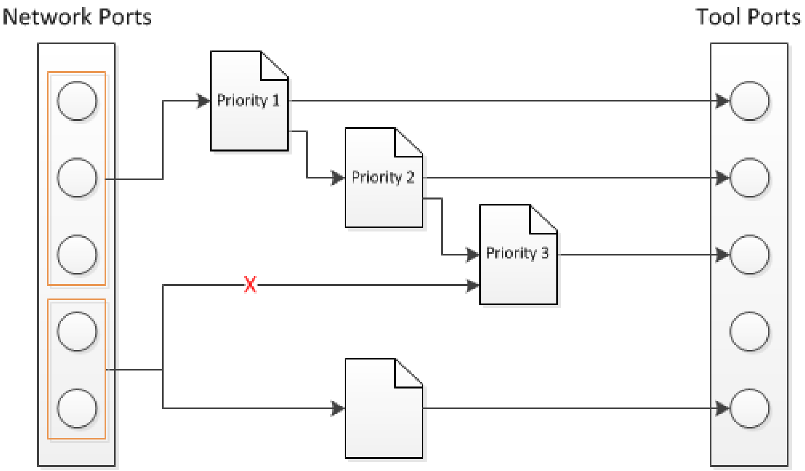

Figure 2-2: Map Chain with Three Maps

Map chains are identified by `mapChainId` attribute of a MapChain object and the priority of a map within a map chain is specified by the `order` attribute of the Map object. Prior to GigaVUE-FM 3.4, the ID of a map chain was specified by the `srcPortsAsId` attribute, which is deprecated.

Working with Rules

This section describes the following:

- *Adding Rules* on page 42
- *Modifying Rules* on page 42
- *Deleting Rules* on page 43

Adding Rules

To add a rule to a map, an application uses the following request:

```
POST <fmip>/api/<api_version>/maps/{alias}
   /rules/{pass|drop}?clusterId=<clusterId>
```

The request contains a payload with the rule to add to the specified map. The following payload example adds an IPv6 rule to a map:

```
{
   "ruleId" : 2,
   "comment" : "",
   "matches" : [ {
   "type" : "ipVer",
      "value" : "v6"
   } ]
}
```

Modifying Rules

To modify a rule in a map, an application uses the following request:

```
PUT <fmip>/api/<api_version>/maps/{alias}/rules/{pass|drop}
   /{ruleId}?clusterId=<clusterId>
```

The request contains a payload with the rule redefinition for the specified map. The following payload example changes the rule IP version to IPv4:

```
{
   "ruleId" : 2,
   "comment" : " ",
   "matches" : [ {
      "type" : "ipVer",
      "value" : "v4"
   } ]
}
```

Deleting Rules

An application can delete all the rules from a map or delete a specific rule.

To delete all rules from a map, an application uses the following request:

```
DELETE <fmip>/api/<api_version>/maps/{alias}/rules/
?clusterId=<clusterId>
```

To delete a specific rule from a map, an application uses the following request:

```
DELETE <fmip>/api/<api_version>/maps/{alias}/rules
   /{ruleId}?clusterId=<clusterId>
```

Working with GigaSMART Groups and Operations

Maps can include GigaSMART operations (gsop) and GigaSMART rules (gsrules). Before an application can add a gsop to a map, the gsop must be in a GigaSMART group (gsgroup). This means that an application may need to create a gsgroup before adding a gsop to a map.

This section describes the following:

- *Creating GigaSMART Groups* on page 43
- *Deleting a GigaSMART Group* on page 43
- *Adding a GigaSMART Operation to a GigaSMART Group* on page 44
- *Getting GigaSMART Operation Information* on page 44
- *Modifying a GigaSMART Operation* on page 44
- *Deleting a GigaSMART Operation* on page 45

Creating GigaSMART Groups

To create a GigaSMART group, an application uses the following request:

```
POST <fmip>/api/<api_version>/gsGroups?clusterId=<clusterId>
```

This request also needs to include a payload that describes the group. For example, the following payload creates a GigaSMART group with the alias GS1 and associates it with engine port 10/1/e1:

```
{
   "alias" : "GS1",
   "ports" : [ "10/1/e1" ],
}
```

Deleting a GigaSMART Group

To delete a GigaSMART group, an application uses the following request:

```
   DELETE <fmip>/api/<api_version>/gsGroups/
{alias}?clusterId=<clusterId>
```

Adding a GigaSMART Operation to a GigaSMART Group

To add a gsop to a GigaSMART group, an application makes the following request:

```
POST <fmip>/api/<api_version>/gsops?clusterId=<clusterId>
```

This request also needs to include a payload that specifies the gsop to add to the gsgroup. For example, the following payload is for a masking operation that is added to the gsgroup with the alias GS1:

```
payload = {
   "alias" : 'masking',
   "gsGroup" : "GS1",
   "gsApps" : {
      "masking" : {
         "protocol" : "none",
         "offset" : 64,
         "pattern" : 'b',
         "length" : 13
      }
   }
}
```

Getting GigaSMART Operation Information

An application can retrieve all gsop in a cluster or retrieve a specific gsop by its alias.

To retrieve all gsop, an application uses the following request:

```
GET <fmip>/api/<api_version>/gsops?clusterId=<clusterId>
```

To retrieve a specific gsop, an application specifies the alias of the gsop in the request:

```
GET <fmip>/api/<api_version>/gsops/gigastreams/
{alias}?clusterId=<clusterId>
```

Modifying a GigaSMART Operation

To modify an existing gsop, an application uses the following request:

```
   PATCH <fmip>/api/<api_version>/gsops/{alias}/
apps?clusterId=<clusterId>
```

For example, to modify the GigaSMART operation added in the previous section, the payload and request are as follows:

```
   payload = {
      "masking" : {
         "protocol" : "tcp",
         "offset" : 4,
         "pattern" : 'b',
         "length" : 13
      }
   }

   PATCH <fmip>/api/<api_version>/gsops/masking/
apps?clusterId=<clusterId>
```

Deleting a GigaSMART Operation

To delete a a gsop, an application uses the following request:

```
DELETE <fmip>/api/<api_version>/gsops/{alias}?clusterId=<clusterId>
```

3 Working with GigaVUE-FM APIs

The GigaVUE-FM APIs can be used as part of IT Operations Management (ITOM) or implemented in a tool to perform a number of tasks. This chapter describes some ITOM use cases. The use cases described are as follows:

- *Working with Nodes* on page 47
- *Working with Ports* on page 53
- *Creating, Modifying, and Deleting Maps* on page 54

Working with Nodes

An application can use the GigaVUE-FM APIs to get nodes, ports, and their status in the Visibility Fabric.

This section describes the following:

- *Adding Nodes* on page 47
- *Getting Node Information* on page 48
- *Getting Chassis Information* on page 51

Adding Nodes

To add nodes, an application does the following:

1. Makes the following request to add nodes or clusters:
   `POST <fmip>/api/<api_version>/nodes`
2. Rediscovers the nodes and clusters with the following request:
   `PUT <fmip>/api/<api_version>/nodes`

The following is a code example that creates three nodes and then rediscovers the nodes managed by GigaVUE-FM after each node is added:

```
nodeAddresses = ["10.115.152.50", "10.115.152.51",
                 "10.115.152.54"]


for address in nodeAddresses:
   payload = {
      "nodeAddSpecs": [{
         "nodeAddress": address,
         "username": "",
         "password": ""
      }]
   }

   POST https://10.115.152.46/api/<api_version>/nodes
   Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
   Content-Type : application/json
   Accept : application/json
   Content-Length : 82

   #On success, make a PUT request to rediscover nodes
   if status = 200:
      PUT https://10.115.152.46/api/<api_version>/nodes
      Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
         Accept : application/json
   else:
      print response.content #If not successful, display response.
      exit()
```

Getting Node Information

The following example requests the information for all nodes as a flat list:

```
GET https://10.115.142.46/api/<api_version>/nodes/flat
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Accept: application/json
```

On success, the response returns the following data in JSON format, showing that there are six nodes. Three nodes are standalone nodes and three nodes belong to a cluster.

```
{
   "nodes" : [ {
      "deviceId" : "10.115.152.50",
      "deviceIp" : "10.115.152.50",
      "dnsName" : "hc2-c04-29.gigamon.com",
      "hostname" : "HC2-C04-29",
      ...
      "clusterMode" : "Standalone",
      "clusterId" : "10.115.152.50",
      ...
   }, {
      "deviceId" : "10.115.152.51",
      "deviceIp" : "10.115.152.51",
      "dnsName" : "ta1-c04-35.gigamon.com",
      "hostname" : "TA1-C04-35",
      ...
      "clusterMode" : "Standalone",
      "clusterId" : "10.115.152.51",
      ...
   }, {
      "deviceId" : "10.115.152.54",
      "deviceIp" : "10.115.152.54",
      "hostname" : "HB1-C03-21",
      ...
      "clusterMode" : "Standalone",
      "clusterId" : "10.115.152.54",
      ...
   }, {
      "deviceId" : "10.115.152.55",
      "deviceIp" : "10.115.152.55",
      "hostname" : "HB1-C03-22",
      ...
      "clusterMode" : "Standalone",
      "clusterId" : "10.115.152.55",
      ...
   }, {
      "deviceId" : "10.115.155.3",
      "deviceIp" : "10.115.155.3",
      "hostname" : "HB1-Team1",
      "family" : "H",
      ...
      "clusterMode" : "Slave",
      "clusterMaster" : "10.115.155.4",
      "clusterId" : "GCC-1",
      ...
   }, {
      "deviceId" : "10.115.155.4",
      "deviceIp" : "10.115.155.4",
      "hostname" : "HC2-Team1",
      ...
      "clusterMode" : "Master",
      "clusterMaster" : "10.115.155.4",
      "clusterId" : "GCC-1",
      ...
   }, {
      "deviceId" : "10.115.155.5",
      "deviceIp" : "10.115.155.5",
      "hostname" : "HD8-Team1",
      ...
      "clusterMode" : "Standby",
      "clusterMaster" : "10.115.155.4",
      "clusterId" : "GCC-1",
      ...
   } ]
}
```

The following example requests the information for all nodes:

```
   GET https://10.115.142.46/api/<api_version>/nodes
   Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
   Accept: application/json
```

On success, the response returns the following data in JSON format, showing the nodes grouped by cluster IDs. The response shows three standalone nodes and a node cluster named GCC-1 with three nodes.

```
{
   "clusters" : [ {
      "family" : "H",
      "clusterId" : "10.115.152.50",
      "members" : [ {
         "deviceId" : "10.115.152.50",
         "deviceIp" : "10.115.152.50",
         "dnsName" : "hc2-c04-29.gigamon.com",
         "hostname" : "HC2-C04-29",
         ...
         "clusterMode" : "Standalone",
         "clusterId" : "10.115.152.50",
         ...
      } ]
   }, {
      "family" : "H",
      "clusterId" : "10.115.152.51",
      "members" : [ {
         "deviceId" : "10.115.152.51",
         "deviceIp" : "10.115.152.51",
         "dnsName" : "ta1-c04-35.gigamon.com",
         "hostname" : "TA1-C04-35",
         ...
         "clusterMode" : "Standalone",
         "clusterId" : "10.115.152.51",
         ...
      } ]
   }, {
      "family" : "H",
      "clusterId" : "10.115.152.54",
      "members" : [ {
         "deviceId" : "10.115.152.54",
         "deviceIp" : "10.115.152.54",
         "hostname" : "HB1-C03-21",
         ...
         "clusterMode" : "Standalone",
         "clusterId" : "10.115.152.54",
         ...
      } ]
   }, {
      "family" : "H",
      "clusterId" : "10.115.152.55",
      "members" : [ {
         "deviceId" : "10.115.152.55",
         "deviceIp" : "10.115.152.55",
         "hostname" : "HB1-C03-22",
         ...
         "clusterMode" : "Standalone",
         "clusterId" : "10.115.152.55",
         ...
      } ]
   }, {
      "family" : "H",
      "clusterId" : "GCC-1",
      "clusterVip" : "10.115.155.2",
      "masterId" : "10.115.155.4",
      "members" : [ {
         "deviceId" : "10.115.155.3",
         "deviceIp" : "10.115.155.3",
         "hostname" : "HB1-Team1",
         ...
         "clusterMode" : "Slave",
         "clusterMaster" : "10.115.155.4",
         "clusterId" : "GCC-1",
         ...
      }, {
         "deviceId" : "10.115.155.4",
         "deviceIp" : "10.115.155.4",
         "hostname" : "HC2-Team1",
         ...
         "clusterMode" : "Master",
         "clusterMaster" : "10.115.155.4",
         "clusterId" : "GCC-1",
         ...
      }, {
         "deviceId" : "10.115.155.5",
         "deviceIp" : "10.115.155.5",
         "hostname" : "HD8-Team1",
         ...
         "clusterMode" : "Standby",
         "clusterMaster" : "10.115.155.4",
         "clusterId" : "GCC-1",
         ...
      } ]
   } ]
}
```

Getting Chassis Information

Queries to the `/node` resource provides only general information about a node or cluster. To get more detailed information, an application can make queries to the resource`/inventory/chassis`.

The following example queries the `/inventory/chassis` resource, and then retrieves card information from the response:

```
GET https://10.115.152.46/api/<api_version>/inventory/
chassis?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Accept : application/json

#On success, display the card information.
if status_code == 200:
   chassisInfo = json.loads(response.text)#Deserialize JSON
   chassisList = chassisInfo['chassisList']
   cards = chassisList[0]['cards']
   for card in cards:
      print 'Slot ID: ' + card['slotId']
      print 'HW Type: ' + card['hwType']
      print 'Operational State: ' + card['operStatus']
      print 'State: ' + card['healthState']
```

Example: Checking for a GigaSMART Card

To use a GigaSMART operation (gsop), a GigaSMART card must be installed. An application can check for a card before adding the operation.

To check for a card and then add a gsop, an application performs the following steps:

1. Makes an inventory of the chassis by sending the following request:

   ```
   GET https://10.115.152.46/api/<api_version>/inventory/
      chassis?clusterId=clusterId=10.115.152.50
   ```

   The response contains a list of the cards.

2. On success of the request (that is, the request returns status code 200), the application looks for the GigaSMART card in the card list.

   The following code example looks for the card in the JSON data returned in the response and prints information about the card:

   ```
   chassisInfo = json.loads(r.text) #Deserialize JSON
   chassisList = chassisInfo['chassisList']
   cards = chassisList[0]['cards']
   for card in cards:
      if card['hwType'] == 'HC2-GigaSMART':
         installed = True
         print 'Slot ID: ' + card['slotId']
         print 'HW Type: ' + card['hwType']
         print 'Amin Status' + card['adminStatus']
         print 'Operational Status: ' + card['operStatus']
         print 'Health: ' + card['healthState']
   ```

3. If the card exists, the application adds the gsop to a gsgroup with the following request:

   ```
   if installed:

      POST https://10.115.152.46/api/<api_version>
         /gsops?clusterId=clusterId=10.115.152.50
   ```

   The following is an example of the payload included in the request to add a gsop for SSL decryption.

   ```
   {
      "alias" : "GSOP_SSL_RSA",
      "gsGroup" : "GS5",
      "gsApps" : {
         "sslDecrypt" : {
            "inPort" : 0,
            "outPort" : 0
         }
      },
      "clusterName" : "10.115.152.50"
   }
   ```

Working with Ports

In addition to using the GigaVUE-FM APIs to configure ports as described in *GigaVUE-FM API Workflows* on page 33 an application can also use the `/inventory/ports` resource to retrieve port information and change port types.

Getting Port Information

An application can use the GigaVUE-FM APIs to get information about ports. The following example inventories the ports and displays the ports with an administrative status of *down*:

```
GET https://10.115.152.46/api/<api_version>/inventory/
   ports?nodeId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json

#On success, display the port information.
if code_status == 200:
   portInventory = json.loads(r.text) #Deserialize JSON.
   ports = portInventory["ports"]
   status = 'down'
   print 'Administrative status ' + status
   for port in ports:
      adminStatus = port['adminStatus']
      operStatus = port ['operStatus']
      if adminStatus == status:
         print 'Port %s' % port['portId']
         print '  Admin Status: %s' % port['adminStatus']
         print '  Op Status: %s' %  port['operStatus']
         print '  Health: %s' % port['healthState']
```

Changing the Port Type and Enabling

An application can use the `/inventory/port` resource to change a port to a different type and change its status, such as change a port to a network port and enable its administrative status.

The following example changes port 5/2/x8 to a network port and port 5/2/x16 to a tool port by using a PATCH operation with `/inventory/ports/{portId}`.

NOTE: When specifying a port ID in a request, replace the forward slash character "/" with the underline character "_". For example, 5/2/x8 becomes 5_2_x8.

```
#List of ports with port type
portList = {'5/2/x8':'network', '5/2/x16':'tool'}
payload = {'portId' : 'id', 'portType" : 'type', 'adminStatus' : 'up'}

#Loop through list of ports
for portId, type in portList.iteritems():

   #Put port ID and port type into payload
   payload["portId"] = portId
   payload["portType"] = type
```

```
#Get content length for request
length = len(json.dumps(payload))

pid = portId.replace['/', '_'] #Change characters in ID.
                               #For example, 5/2/x8 becomes 5_2_x8.

PATCH https://10.115.152.46/api/<api_version>/inventory/ports/
      pid?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
Content-Length : length

#Display changes to port if PATCH request succeeds
if status_code = 200:

   GET https://10.115.152.46/api/<api_version>/inventory/ports/
      pid?clusterId=10.115.152.50
   Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
   Accept : application/json

   portInventory = json.loads(r.text)#Deserialize JSON
   port = portInventory["port"]
   print 'Port %s' % port['portId']
   print '  Port Type: %s' % port['portType']
   print '  Admin Status: %s' % port['adminStatus']
   print '  Op Status: %s' %  port['operStatus']
else:
   #Handle error if return status is not 200
   gigaErrors = json.loads(resp.text)#Deserialize JSON
   errors = gigaErrors['errors']
   for err in errors:
      print 'GigaError ' +  err['code']
      print err['msg']
      exit()
```

Creating, Modifying, and Deleting Maps

This section provides information about how an application can use the GigaVUE-FM APIs to work with maps. This section covers the following topics:

- *Creating Maps* on page 55
- *Modifying Maps* on page 60
- *Deleting Maps* on page 62

Creating Maps

The following section shows some examples of creating maps with the GigaVUE-FM APIs. The examples are as follows:

- *Creating a Map By Rule* on page 55

  In this example, a map by rule is created, which is later used in the section *Modifying Maps* on page 60.

- *Creating Maps for Dropping Traffic on a Session* on page 56

  In this example, a first-level, a second-level, and a collector map are created.

- *Creating Maps for Masking Data* on page 58

  In this example, first- and second-level maps are created.

Creating a Map By Rule

The following example creates a map by rule that passes IPv4 and IPv6 traffic between the source network port 5/2/x8 and the destination tool port 5/2/x16. An application can set the port types of these ports and their administrative status as described in *Changing the Port Type and Enabling* on page 53 before creating the map.

```
payload = {
      'alias': 'Map_RSA_Traffic',
      'rules': {
      'dropRules': [],
      'passRules': [{
         'comment': '',
         'matches': [{
         'type': 'ipVer',
         'value': 'v4'
      }],
      'ruleId': 1
   },{
      'comment': '',
      'matches': [{
         'type': 'ipVer',
         'value': 'v6'}],
      'ruleId': 2
   ]},
   'subType': 'byRule',
   'clusterName': '10.115.152.50',
   'srcPorts': ['5/2/x8'],
   'ruleMatching': 'normal',
   'type': 'regular',
   'order': 1,
   'dstPorts': [ '5/2/x16' ]
}


POST https://10.115.152.46/api/<api_version>/
maps?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
```

```
Content-Length : 384
```

Creating Maps for Dropping Traffic on a Session

This section shows an example of how to use the `/map` APIs to create maps that drop traffic for a session. In this use case, an application does the following:

1. Creates a first-level map that passes any MAC address and connects the map to a network port and a virtual port.
2. Creates a second level map that implements a Session-aware Adaptive Packet Filtering (SAPF) gsop and connects this map to the virtual port in the previous step and a tool port.
3. Creates a collector map connected to the same virtual port and tool port as the second level map.

Figure 3-1 shows the maps and their relation to the ports resulting from the previous three steps.

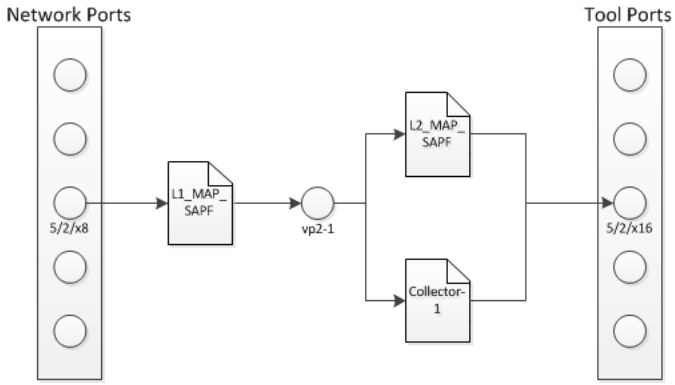

Figure 3-1: Maps for Dropping Traffic

Creating the First-level Map

The following example creates a first-level map that passes any MAC address. The source port (`srcPorts`) is 5/2/x8. The destination port (`dstPorts`) is a virtual port, vp2-1.

```
payload = {
      "alias": "L1_Map_SAPF",
      "rules": {
         "passRules": [{
            "comment": " ",
            "ruleId": 1,
            "matches": [{
               "type": "macSrc",
               "mask": "0000.0000.0000",
               "value": "0000.0000.0000"
            }]
         }]
      },
      "dstPorts": [ "vp2-1" ],
      "subType": "byRule",
      "srcPorts": [ "5/2/x8" ],
      "type": "firstLevel",
      "order": 1
   }

POST https://10.115.152.46/api/<api_version>/
maps?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
Content-Length : 263
```

Creating the Second-level Map

The following example shows how to create a second-level map with a SAPF gsop. The source port (`srcPorts`) is vp2-1. The destination port (`dstPorts`) is 5/2/x16.

```
payload = {
      "alias": "L2_Map_SAPF",
      "srcPorts": [ "vp2-1" ],
      "subType": "byRule",
      "dstPorts": [ "5/2/x16" ],
      "type": "secondLevel",
      "gsop": "GSOP_GS2_APF",
      "gsRules": {
         "dropRules": [{
            "matches": [{
               "matchOffset": {
                  "offsetStart": 1,
                  "offsetEnd": 1750,
                  "protocol": {
                     "protocol": "tcp",
                     "pos": 1
                  }
               },
               "value": "netflix|nflxvideo|nflximg|Netflix|nflxext",
               "matchType": "regex",
               "type": "pmatch"
            }],
            "ruleId": 1
         }]
      }
   }

POST https://10.115.152.46/api/<api_version>/
maps?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
Content-Length : 376
```

Creating the Collector Map

The following example creates a shared collector map. The source port is (`srcPorts`) is vp2-1. The destination port (`dstPOrts`) is a virtual port, 5/2/x16.

```
payload = {
      "alias": "Collector-1",
      "dstPorts": [ "5/2/x16" ],
      "subType": "collector",
      "srcPorts": [ "vp2-1" ],
      "type": "secondLevel"
   }
POST https://10.115.152.46/api/<api_version>/
maps?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
Content-Length : 119
```

Creating Maps for Masking Data

Often certain fields must be masked with a pattern to protect sensitive information during network analysis. In this use case, an application creates maps that mask the data when the data matches a certain pattern. To create the maps and mask the data, an application performs the following steps:

1. Creates a first-level map that is connected a network port and a virtual port. This map passes any MAC address.
2. Creates a second-level map that is connected to the virtual port in step 1 and a tool port. This second-level map implements a masking gsop.

Creating the First-level Map

The following example creates a first-level map that passes any MAC address. The source port (`srcPorts`) is 5/2/x8. The destination port (`dstPorts`) is a virtual port, vp5-1.

```
payload = {
      "alias": "L1_Map_Mask",
      "rules": {
         "passRules": [{
            "comment": "",
            "ruleId": 1,
            "matches": [{
               "type": "macSrc",
               "mask": "0000.0000.0000",
               "value": "0000.0000.0000"
            }]
         }]
      },
      "dstPorts": [ "vp5-1" ],
      "subType": "byRule",
      "srcPorts": [ "5/2/x8" ],
      "type": "firstLevel",
      "order": 1
```

```
   }

POST https://10.115.152.46/api/<api_version>/
maps?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
Content-Length : 264
```

Creating the Second-level Map

The following example creates a second-level map with an APF gsop and masking, where 0 replaces the numbers in an identification number. The source port (`srcPorts`) is vp2-5. The destination port (`dstPorts`) is 5/2/x16.

```
payload = {
      "alias": "L2_Map_Mask",
      "srcPorts": [ "vp5-1" ],
      "subType": "byRule",
      "dstPorts": [ "5/2/x16" ],
      "type": "secondLevel",
      "gsop": "GSOP_GS5_APF_Mask",
      "gsRules": {
         "passRules": [{
            "matches": [{
               "matchOffset": {
                  "offsetStart": 1,
                  "offsetEnd": 1750,
                  "protocol": {
                     "pos": 1,
                     "protocol": "tcp"
                  }
               },
               "mask": {
                  "pattern": "0",
               },
               "value": "\\d{3}-?\\d{7}-?\\d{7}",
               "matchType": "regex",
               "type": "pmatch"
            }],
            "ruleId": 2
         }]
      }
   }

POST https://10.115.152.46/api/<api_version>/
maps?clusterId=10.115.152.50
Authorization : Basic 'YWRtaW4gYWRtaW4xMjNBIQ=='
Content-Type : application/json
Accept : application/json
Content-Length : 389
```

Modifying Maps

An application can use the GigaVUE-FM APIs to modify a map when a tool detects an event in the traffic flow. In the use case described in this section, traffic is flowing between two hosts after a flow map was created. The tool detects SSL traffic with clear text. However, the tool that is monitoring the traffic needs the flow decrypted. It indicates to the application that it needs to add an SSL decryption GSOP to the existing map. Figure 3-2 illustrates the process.

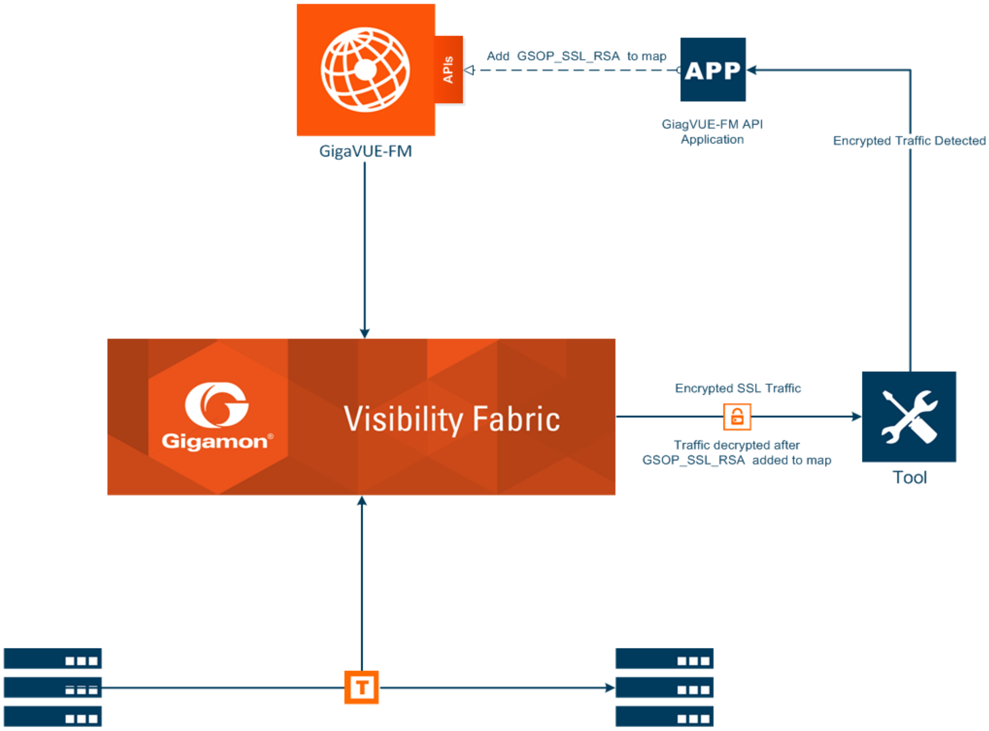
APIs
Add GSOP_SSL_RSA to map
APP
GigaVUE-FM
GiagVUE-FM API Application
Encrypted Traffic Detected
Gigamon
Visibility Fabric
Encrypted SSL Traffic
Traffic decrypted after GSOP_SSL_RSA added to map
Tool
T

Figure 3-2: GigaVUE-FM API Application Modifying a Map

To modify the map, the application uses a PUT operation to modify the map, specifying the map by its alias. In this example, the map Map_RSA_Traffic previously described in *Creating a Map By Rule* on page 55 is modified.

The following is an example of a request and payload that the application can send to GigaVUE-FM to modify the map with the alias Map_RSA_Traffic. The request adds the SSL RSA gsop, changes the order to 2 from 1, and the IP from IPv6 to IPv4.

```
payload = {
   "alias" : "Map_RSA_Traffic",
   "clusterName" : "10.115.152.50",
   "type" : "regular",
   "subType" : "byRule",
   "srcPorts" : [ "5/2/x8" ],
   "dstPorts" : [ "5/2/x16" ],
   "gsop" : "GSOP_SSL_RSA",
   "order" : 2,
   "rules" : {
      "dropRules" : [ ],
      "passRules" : [ {
         "ruleId" : 1,
         "comment" : "",
         "matches" : [ {
            "type" : "ipVer",
            "value" : "v4"
         } ]
      } ]
   },
   "ruleMatching" : "normal"
}
```

```
PUT https://10.115.152.46/api/<api_version>/maps/
Map_RSA_Traffic?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Content-Type : application/json
Accept : application/json
Content-Length : 340
```

By making a GET request to the `/map` resource, the application shows that the map now includes the SSL RSA gsop and the other changes.

```
Get https://10.115.152.46/api/<api_version>/maps/
   Map_RSA_Traffic?clusterId=10.115.152.50
Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
Accept : application/json
```

The following is the content in the response:

```
{
   "map" : {
      "alias" : "Map_RSA_Traffic",
      "clusterId" : "10.115.152.50",
      "type" : "regular",
      "subType" : "byRule",
      "srcPorts" : [ "5/2/x8" ],
      "dstPorts" : [ "5/2/x16" ],
      "gsop" : "GSOP_SSL_RSA",
      "order" : 2,
      "rules" : {
         "dropRules" : [ ],
         "passRules" : [ {
            "ruleId" : 1,
            "comment" : "",
            "bidi" : false,
            "matches" : [ {
               "type" : "ipVer",
               "value" : "v4"
         } ]
      } ]
   },
      "roles" : {
         "owners" : [ "admin" ],
         "viewers" : [ ],
         "editors" : [ ],
         "listeners" : [ ]
      },
      "ruleMatching" : "normal"
   }
}
```

Deleting Maps

To delete maps, an application uses the DELETE operation on the `/maps` resource. The following example deletes three maps by their alias:

```
mapList = ["L1_Map_SAPF", "L1_Map_SAPF", "Collector-1"]
for mapAlias in mapList:
   DELETE https://10.115.152.46/api/<api_version>/maps/mapAlias
         ?clusterId=10.115.152.50
   Authorization: Basic 'dXNlcm5hbWU6cGFzc3dvcmQ='
   Accept : application/json

   #Display status code and message if request does not delete map.
   if code_status != 200:
      print 'Request failed...'
      gigaErrors = json.loads(resp.text)
      errors = gigaErrors['errors']
      for err in errors:
         print 'GigaError ' +  err['code']
         print err['msg'] + '\n'
         exit()
```

4 GigaVUE-FM API Reference

The the GigaVUE-FM API Reference is available as part of the GigaVUE-FM Online Help. To view the reference, select **API Reference** in the Navigation pane under Support.

4414-02 12/16