Illumio Core®

Compatible PCE Versions: All PCE releases

FlowLink

Version: 1.1.2

FlowLink Configuration and Usage Guide

May 2022

26000-100-1.1.2

Legal Notices

Product Versions

FlowLink Version: 1.1.2

Compatible PCE Versions: All PCE releases

Standard versus LTS Releases

For information on Illumio software support for Standard and LTS releases, see Versions and Releases on the Illumio Support portal.

Resources

Legal information, see https://www.illumio.com/legal-information

Trademarks statements, see https://www.illumio.com/trademarks

Patent statements, see https://www.illumio.com/patents

License statements, see https://www.illumio.com/eula

Open source software utilized by the Illumio Core and their licenses, see *Open Source Licensing Disclosures* in the Illumio Core Technical Documentation portal.

Contact Information

To contact Illumio, go to https://www.illumio.com/contact-us

To contact the Illumio legal team, email us at legal@illumio.com

To contact the Illumio documentation team, email us at doc-feedback@illumio.com

Contents

Chapter 1

About FlowLink

This chapter contains the following topics:

This section describes the FlowLink application, the types of flow records it supports, it's scale, and limitations.

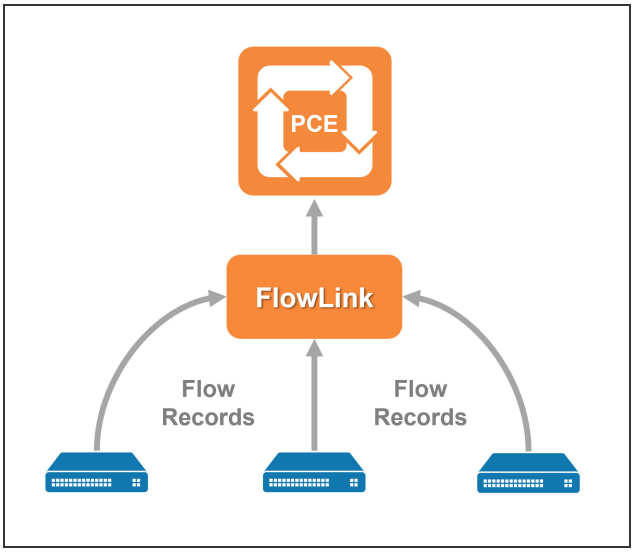
Overview

The FlowLink application normalizes and aggregates the network flow data that it collects from different types of network sources into a format that can be ingested by the PCE for use by traffic data applications. It does not resolve any flow data source and destination IP addresses in to the PCE workloads. The PCE displays the flow in Illumination and marks the policy decision as 'unknown'. FlowLink is supported on standard PCE clusters and also on Super-cluster.

How FlowLink Works

FlowLink can receive the flow data by connecting to a data source provided by you and adheres to your organizations' data format. It may consume flows at a rate that is slower than the source speed. Therefore, the flow sender caches the flow data for 48 hours or more. If the PCE is unable to accept flow data because of the rate of flow or availability issues, FlowLink caches the data

locally to a disk for a configurable period of time or disk space and retries periodically (user-configurable number of minutes). It aggregates data flows and sends them to the PCE once every configurable number of minutes. It does not have access to the PCE data and therefore no knowledge of workloads, virtual services, and other objects.



NOTE:
FlowLink version 1.1.0 does not support a High Availability (HA) configuration. You will have to monitor FlowLink and ensure that you restart it on failure.

Supported Flow Record Formats

The following types of flow records are supported:

- AWS VPC flows
- IPFIX v10
- NetFlow v5, v7, v9, and v10
- sFlow v5
- Text (customizable parser configured by user, for example, Syslog or Kafka)

Scale and Limitations

This section lists the supported scale and known limitations to be considered while using FlowLink.

PCE

- The PCE processes up to 10K unique flows/second. This is the total number of FlowLink and VEN flows received by the PCE.
- The PCE handles up to 20 concurrent POSTs.
- The PCE allows a maximum file size of 100MB per POST.
- For each IP address that exists in your data flows, you need to create corresponding unmanaged workloads in the PCE, if you want to see those traffic flows in Illumination. Else, those flows will not be displayed.

FlowLink

- FlowLink supports multiple flow data sources.
- The maximum number of sources per FlowLink are not reported. For best practices, consider one source per FlowLink.
- Flows with Class D addresses are ignored.
- The following two limitations are generic traffic limitations with Illumination and are not specific to FlowLink:
    - At least one IP address in the reported flow must match to an IP address of a workload object (managed or unmanaged).
    - If a virtual service object and workload object have the same IP address, then flow lines will always be drawn to the virtual service.

Chapter 2

FlowLink Configuration

This chapter contains the following topics:

This section describes how to configure and run FlowLink.

Configure FlowLink

This section provides requirements and steps you need to follow to configure FlowLink.

Requirements

- CentOS or RHEL server
- Root privileges to the server
- FlowLink RPM downloaded from the Illumio Support site
- PCE with API Key and Secret

IMPORTANT:
Role! You must have Global Administrator or Global Organization Owner privileges.

CPU, Memory, and Storage Requirements

To install FlowLink, your hardware must meet the capacity requirements detailed in this section.

Machine Type	Cores/Clock Speed[1]	RAM per Node[2]	Storage Device Size[3] and IOPS[4]
FlowLink 2500 work-loads	• 2 cores • Intel® Xeon(R) CPU E5-2695 v4 at 2.10GHz or equivalent	8 GB	• 1 x 20 GB • 100 IOPS per device

Footnotes:

[1] CPUs:

- The recommended number of cores is based only on physical cores from allocated CPUs, irrespective of hyper-threading or virtual cores. For example, in AWS one vCPU is only a single hyper-thread running on a physical core, which is half a core. 16 physical cores equates to 32 vCPUs in AWS.
- Full reservations for vCPU. No overcommit.

[2] Full reservations for vRAM. No overcommit.

[3] Additional disk notes:

- Storage requirements for network traffic data can increase rapidly as the amount of network traffic increases. Allocating a separate, large storage device for traffic data can accommodate these rapid changes without potentially interrupting the service.
- Network File Systems (NFS) is not supported.

[4] Input/output operations per second (IOPS) are based on 8K random write operations. IOPS specified for an average of 300 flow summaries (80% unique `src_ip, dest_ip, dest_port, proto`) per workload every 10 minutes. Different traffic profiles might require higher IOPS.

FlowLink Storage Partitioning

Storage Device	Partition mount point	Size to Allocate	Notes
Device 1, Partition A	/	20 GB	Logrotate must be configured to limit the disk consumption of Flow & System Logs.

Install FlowLink RPM

1. Login as a root user.
2. Install the RPM.
   The default install location is: `/usr/local/bin/`

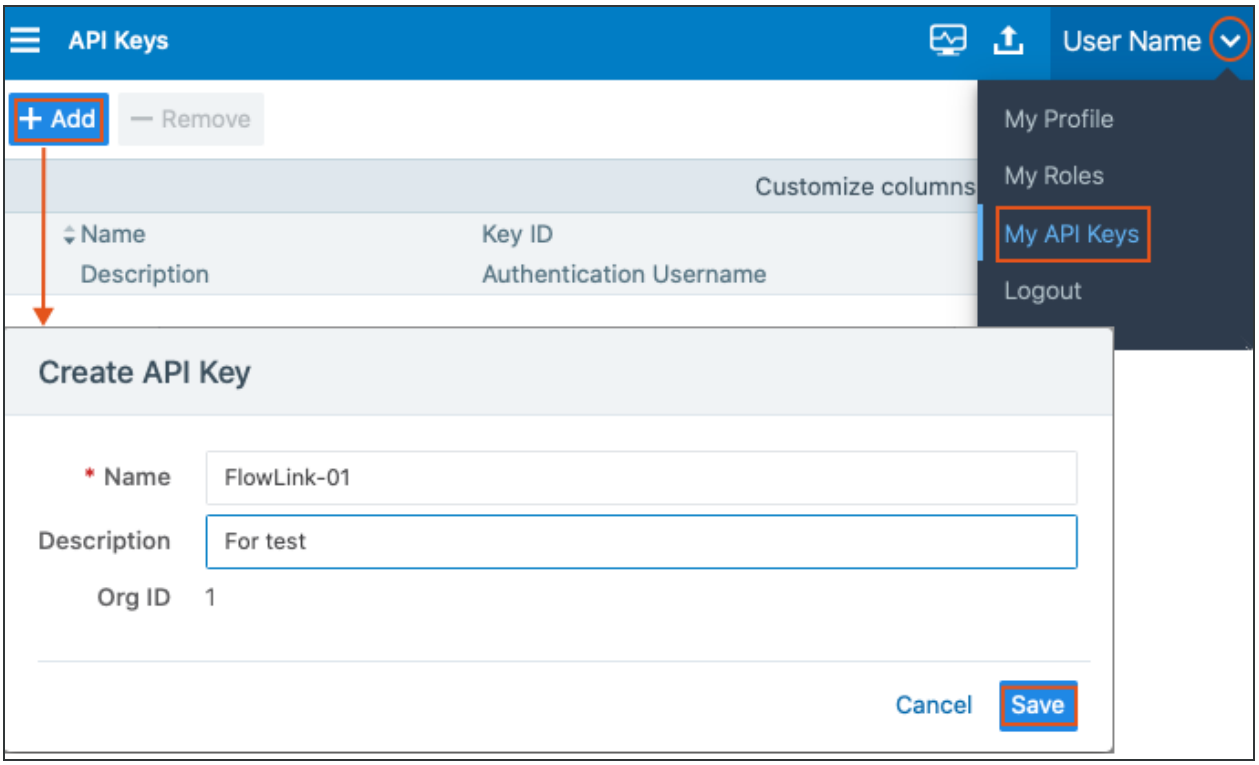
```
sudo su
rpm -ivh illumio-flowlink-1.1.0-45.x86_64.rpm
```

IMPORTANT:
Login! Only the Install FlowLink RPM step needs root user login. The Create PCE API File, Create YAML Configuration File, and Run FlowLink steps can be run by logging in as any user.

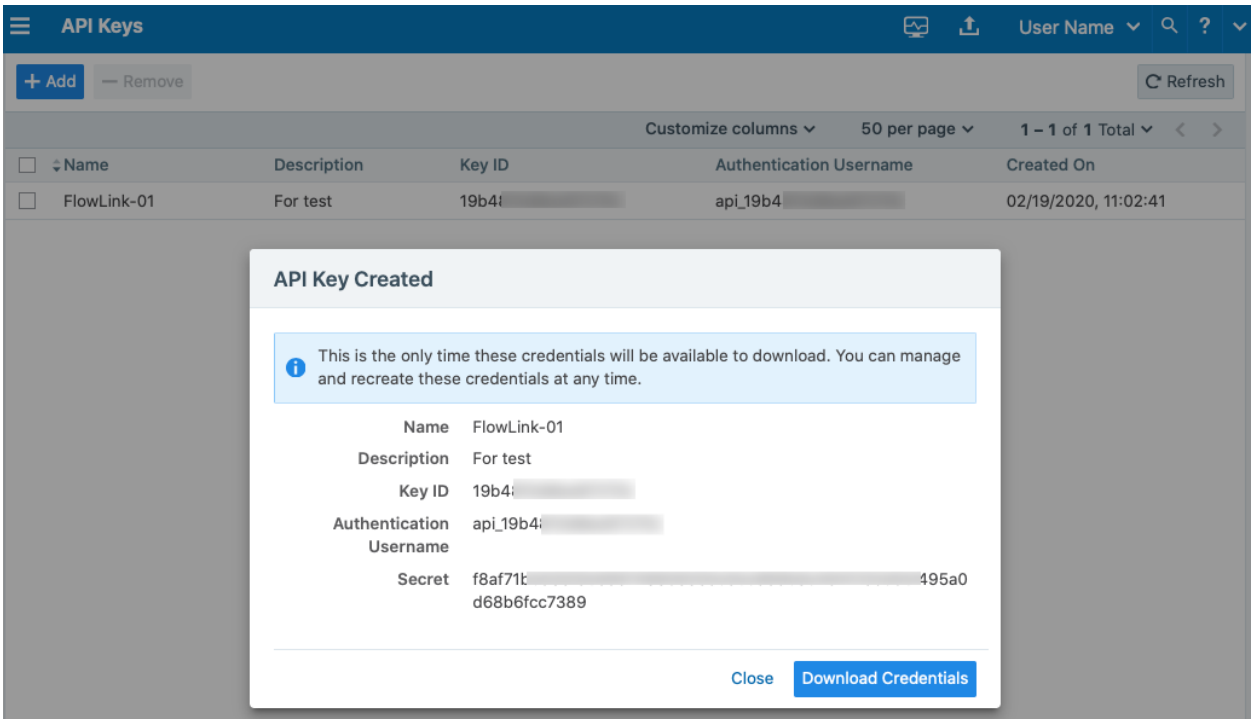
In the following sections `/home/employee` directory is used as an example. The **api_info** file should be in a directory writable by the user, for example in the `/home/employee` directory.

Create PCE API File

1. To generate an API key, click **My API Keys** from the upper-right corner drop-down menu in the PCE UI.
2. The 'API Keys' page opens. Click **Add**.
3. The 'Create API Key' page opens. Enter a Name (mandatory) and Description (optional) and click **Save**.



4. The API Key is created.



5. Copy the values of the 'Authentication Username' and 'Secret' in to a text file on the FlowLink server.
   Use a space to separate the key and secret. For example:
   `api_xxxxxxxxxxxxx yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy`
6. Copy the absolute path of the file PCE API file `/home/employee/api_info`.
   You will need it in the FlowLink configuration file.

Create YAML Configuration File

1. In the `/home/employee` directory, create a YAML configuration file. You can find an example yml file at `/usr/local/illumio/config.yml.example`.
2. Enter the parameters. For more details about parameters, see Key Value Parameters.
   Example of FlowLink configuration:

```
pce_addr: mypce.example.com:8443
api_key: $cat /home/employee/api_info
data_directory: /home/employee
```

```
aggregation_minutes: 10
consumers:
  - name: netflow
    parser:
      type: netflow
    connectors:
      - type: udp
        properties:
          ports: '2055'
```

The above configuration listens for NetFlow on UDP 2055 from any data source. The absolute path is: `/home/employee/config.yaml.netflow`

Run FlowLink

1. To manage FlowLink, use the following commands:

   ```
   illumio-flowlink-ctl start --config <path to config file> [--log-file <path
   to log file>]
   illumio-flowlink-ctl stop
   illumio-flowlink-ctl status
   ```

   The default path for the log file is <data_directory specified in config file>/`flowlink.log`

2. To start FlowLink, use the `illumio-flowlink-ctl start` command. Make sure that you include the `--config` option in the start command, which will begin running the program in the background.

   Example with expected output:

   ```
   illumio-flowlink-ctl start --config /home/employee/config.yaml.netflow

   OUTPUT TO CONSOLE
   Checking Flowlink started successfully.
   OK.
   Output logs can be found at: /home/employee/flowlink.log
   ```

```
OUTPUT IN LOG FILE (/home/employee/flowlink.log)
2020-03-11T09:58:51.173203-07:00 Waiting for signal
2020-03-11T09:58:51.330757-07:00 Starting Data Consumer: netflow
2020-03-11T09:58:51.331162-07:00 Listening for netflow messages on udp port:
2055
2020-03-11T09:58:51.332929-07:00 Reporting flows every 10 minutes
```

3. To stop FlowLink, use the `illumio-flowlink-ctl stop` command.

   Example with expected output:

```
illumio-flowlink-ctl stop

OUTPUT ON CONSOLE
/illumio-flowlink-ctl stop
Stopping FlowLink: ......... Stopped.

OUTPUT IN LOG FILE (/home/employee/flowlink.log)
2020-03-11T09:58:57.097817-07:00 Got signal
2020-03-11T09:58:57.097835-07:00 Telling connectors to stop
2020-03-11T09:58:57.097856-07:00 Allowing parsers to drain
2020-03-11T09:58:57.098766-07:00 udp exiting
2020-03-11T09:58:57.098800-07:00 udp exiting
2020-03-11T09:58:57.101361-07:00 udp exiting
2020-03-11T09:58:57.101400-07:00 udp exiting
2020-03-11T09:58:57.103881-07:00 udp exiting
2020-03-11T09:58:57.103905-07:00 udp exiting
2020-03-11T09:58:57.106527-07:00 udp exiting
2020-03-11T09:58:57.106579-07:00 udp exiting
2020-03-11T09:58:57.109120-07:00 udp exiting
2020-03-11T09:58:57.109145-07:00 udp exiting
2020-03-11T09:58:57.111790-07:00 udp exiting
2020-03-11T09:58:57.111837-07:00 udp exiting
2020-03-11T09:58:57.113853-07:00 udp exiting
2020-03-11T09:58:57.113912-07:00 udp exiting
2020-03-11T09:58:57.116262-07:00 udp exiting
2020-03-11T09:58:57.116397-07:00 udp exiting
```

```
2020-03-11T09:58:57.118365-07:00 udp exiting
2020-03-11T09:58:57.119002-07:00 udp exiting
2020-03-11T09:58:57.120865-07:00 udp exiting
2020-03-11T09:58:57.121108-07:00 udp exiting
2020-03-11T09:58:57.123517-07:00 udp exiting
2020-03-11T09:58:57.123552-07:00 udp exiting
2020-03-11T09:58:57.126043-07:00 udp exiting
2020-03-11T09:58:57.126079-07:00 udp exiting
2020-03-11T09:59:02.100923-07:00 Writing flows
2020-03-11T09:59:02.100969-07:00 Flow count: 48468
2020-03-11T09:59:02.417261-07:00 Waiting for file senders to drain
2020-03-11T09:59:02.418564-07:00 Sending file: /home/employee/traffic_flows_
1583945942416835.pb.gz
2020-03-11T09:59:07.390307-07:00 Response Code 204
```

4. To check the status of FlowLink, use the `illumio-flowlink-ctl status` command.

   Example with expected output:

```
illumio-flowlink-ctl status

OUTPUT ON CONSOLE
/illumio-flowlink-ctl status
FlowLink: RUNNING
```

Configure YAML

FlowLink requires configurable parameters using a YAML file.

NOTE:
Refer to the `/usr/local/illumio/flowlink_config_schema.json` file provided with the FlowLink RPM for definitions of all the fields supported by the FlowLink configuration file.

Key Value Parameters

This table describes the YAML file key-value parameters.

Parameter	Required/Optional	Description
`aggregation_minutes`	Optional	The interval (in minutes) in which flows are aggregated and sent to the PCE. Default interval: 10 Minimum allowed interval: 5 Maximum allowed interval: 60 For example: `aggregation_minutes: 10`
`api_key`	Required	API key and secret of the PCE. This allows FlowLink to POST flows to the PCE. The API key and secret can be copied into a file. You can run a script to *cat* the contents of that file. In the example below, a file called *api_info* is created which contains the PCE API key and secret. For example: `api_key: $cat /home/em-ployee/api_info`
`consumers`	Required	A list of dictionaries. It requires a name, parser, and connector. FlowLink configuration sup-ports one or many consumers (flow types). For more details about con-figuring the ingested flow

Parameter	Required/Optional	Description
		types, see Ingested Flow Types.
`data_directory`	Required	The pathname of a directory where FlowLink can store any unsent data flow files or any restart information. For example: `data_directory: /home/employee/`
`data_directory_size_mb`	Optional	The maximum size (in Mega-bytes) of data that can be stored in the data directory before being pruned. Default: 500 Minimum value: 100 For example: `data_directory_size_mb: 200`
`file_retention_hours`	Optional	The maximum number of hours unsent data flow files will be stored before being pruned. Default: 24 Minimum: 4 For example: `file_retention_hours: 8`
`metrics_print_seconds`	Optional	The frequency (in seconds) at which the metrics information is printed. Default: 60 Minimum: 15 For example: `metrics_print_seconds: 60`

Parameter	Required/Optional	Description
`org_id`	Required for SaaS Optional for on-premises	The org id to which the flow data will be posted. The default id is 1. For example: `org_id: 1`
`pce_addr`	Required	FQDN of the PCE and port. For example: `pce_addr: https://mypce.example.com:8443`

Ingested Flow Types

This section provides the Consumer Syntax when using various supported parsers and connectors.

IPFIX, NetFlow, and sFlow Parsers

```
consumers:
  - name:  # Required. An array of properties defining the data consumers
configured for FlowLink. For example: netflow
    parser:
      type: #Required. Information describing the parser associated with the data
consumer. List of supported values: 'netflow', 'ipfix', 'sflow', 'aws', or 'text'
    connectors:
      - type: #Required. Information describing the data source connector
associated with the data consumer. Supported values: 'udp', 'tcp', 'kafka', or
'aws'
        properties:
          ports: #Required parameter to describe tcp or udp port. For example:
'2055'
          remote_addrs: #Optional parameter. String or list of IP address(es) to
listen for as trusted data sources. Default is allow all IPs. CIDRs are not
supported. For example: '192.168.1.10,192.168.1.15'.
```

AWS Parser and Connector

```
consumers:
  - name:  # Required. An array of properties defining the data consumers
configured for FlowLink. For example: aws
    parser:
      type: #Required. Information describing the parser associated with the data
consumer. Supported value: aws
    connectors:
      - type: #Required. Information describing the data source connector
associated with the data consumer. Supported value: aws
        properties:
          region: #Required. Configures the AWS region of where the VPC flow logs
are stored. Value not wrapped in quotes. Examples: us-west-2 or us-east-1
          credentials: #Required. This is the AWS Access Key ID and AWS Access Key
Secret created by IAM. The IAM user must have privileges to read Cloud Watch logs.
You can put the contents into a file and run a script to cat the file. Value not
wrapped in quotes. For example: $cat /home/employee/aws_info
          log_groupname: #Required. The name of the AWS Log Group. Value not
wrapped in quotes. For example: myVPCFlowLogs
```

NOTE:
The Access Key ID and Key Secret format should be the same as defined in YAML Configuration.

Text Parser with TCP or UDP Connector

```
consumers:
  - name: # Required. An array of properties defining the data consumers
configured for FlowLink. For example: syslog
    parser:
      type: #Required. Information describing the parser associated with the data
consumer. Supported value: 'text'
      properties:
        src_ip: #Required. Attribute tag or field number (starting at 1) used to
extract source IP. For example: sip
        dst_ip: #Required. Attribute tag or field number (starting at 1) used to
```

```
extract destination IP. For example: dip
        dst_port: #Required. Attribute tag or field number (starting at 1) used to
extract destination port. For example: dport
        protocol: #Required. Attribute tag or field number (starting at 1) used to
extract protocol. For example: prot
        icmp_type: #Optional. Attribute tag or field number (starting at 1) used
to extract icmp type. For example: type
        icmp_code: #Optional. Attribute tag or field number (starting at 1) used
to extract icmp code. For example: code
        timestamp: #Optional. Attribute tag or field number (starting at 1) used
to extract timestamp. Default: 1. For example: "date_time, 1"
        timestamp_format: #Optional. A string used to describe the timestamp
format field(s) in a record. The following values can be used year: yy[yy], month
(Jan[uary] etc): mmm[mmm], dayOfMonth: dd or _d, dayOfWeek(Mon[day], etc): ddd
[ddd], hour: HH, minutes: MM, seconds(with optional precision): SS[.0{1 or more}],
timeZone: ZZZ, -HH[:MM], -HHMM, ZHH[:MM], ZHHMM, unix timestamp: unix. For
example: "mm dd yyyy HH:MM:SS"
    connectors:
      - type: #Required. Information describing the data source connector
associated with the data consumer. List of supported values: 'tcp', 'udp', or
'sctp'
        properties:
          ports: #Required parameter to describe tcp or udp port. For example:
'514'
          remote_addrs: #Optional. A comma separated list of remote host addresses
from which to accept flows. For example: '192.168.200.13'
```

Text Parser with Kafka Connector

```
consumers:
  - name: # Required. An array of properties defining the data consumers
configured for FlowLink. For example: syslog
    parser:
      type: #Required. Information describing the parser associated with the data
consumer. Supported value: 'text'
      properties:
        src_ip: #Required. Attribute tag or field number used to extract source
IP. For example: sip
```

```
        dst_ip: #Required. Attribute tag or field number used to extract
destination IP. For example: dip
        dst_port: #Required. Attribute tag or field number used to extract
destination port. For example: dport
        protocol: #Required. Attribute tag or field number used to extract
protocol. For example: prot
        icmp_type: #Optional. Attribute tag or field number used to extract icmp
type. For example: type
        icmp_code: #Optional. Attribute tag or field number used to extract icmp
code. For example: code
        timestamp: #Optional. Attribute tag or field number used to extract
timestamp. For example: "date_time, 1"
        timestamp_format: #Optional. A string used to describe the timestamp
format  field(s) in a record. The following values can be used year: yy[yy], month
(Jan[uary] etc): mmm[mmm], dayOfMonth: dd or _d, dayOfWeek(Mon[day], etc): ddd
[ddd], hour: HH, minutes: MM, seconds(with optional precision): SS[.0{1 or more}],
timeZone: ZZZ, -HH[:MM], -HHMM, ZHH[:MM], ZHHMM, unix timestamp: unix. For
example: "mm dd yyyy HH:MM:SS"
    connectors:
      - type: kafka
        properties:
          version: #Required. The version of the kafka broker(s). For example:
1.2.0
          brokers: #Required. A comma separated list of kafka brokers using FQDN
and port. For example: example.com:9092
          group: test
          topics: test
          client_id: flowlink
```

Ingested Flow Examples

This section provides flow examples while using the supported parsers and connectors.

IPFIX

The below example shows a consumer that listens for IPFIX on UDP 4739 coming only from an IPFIX exporter whose IP address is 192.168.11.5. The flows from other IPFIX exporters will be discarded.

```
consumers:
  - name: ipfix
    parser:
      type: ipfix
    connectors:
      - type: udp
        properties:
          ports: '4739'
          remote_addrs: '192.168.11.5'
```

NetFlow

The below example is using NetFlow in which FlowLink will parse NetFlow records via UDP 6500 and listen for any data source IP address.

```
consumers:
  - name: netflow
    parser:
      type: netflow
    connectors:
      - type: udp
        properties:
          ports: '6500'
```

AWS

The below example is of an AWS consumer in which the CloudWatch Log Group name is myVPCFlowLogs and is configured in the AWS Oregon region.

```
consumers:
  - name: aws
    parser:
      type: aws
    connectors:
      - type: aws
        properties:
          region: us-west-2
```

```
          credentials: $cat /home/employee/aws_info
          log_groupname: myVPCFlowLogs
```

Text

The below example is of a text consumer using Syslog and listening on UDP 6514. The syslog format uses sip attribute to extract the source IP of the flow.

```
consumers:
  - name: syslog
    parser:
      type: text
      properties:
        src_ip: sip
        dst_ip: dip
        dst_port: dport
        protocol: prot
        timestamp: "date_time, 1"
        timestamp_format: "mmm dd yyyy HH:MM:SS"
    connectors:
      - type: udp
        properties:
          ports: "6514"
```

YAML

```
pce_addr: 2x2mypce.example.com:8443
api_key: $cat api_info
data_directory: /home/employee/
aggregation_minutes: 5
consumers:
  - name: netflow
    parser:
      type: netflow
    connectors:
      - type: udp
```

```
      properties:
        ports: '6500'
- name: ipfix
  parser:
    type: ipfix
  connectors:
    - type: udp
      properties:
        ports: '6514'
```

Chapter 3

FlowLink Usage

This chapter contains the following topics:

This section describes how to export IPFIX or NetFlow v9 flow records from F5 BIG-IP to an external flow collector and some solutions while troubleshooting.

Collect Flow Records from F5

The example listed in the following steps uses a virtual edition of the F5 BIG-IP appliance in AWS and the Illumio FlowLink application to gather and parse flow data.

IMPORTANT:
IPFIX and NetFlow have slightly different configuration steps depending on which flow record standard you choose.

Requirements

- FlowLink (flow collector)
- F5 BIG-IP system with LTM
- A virtual server configured on F5 box

NOTE:
F5 must have a self-IP interface. The flows are sent out of this interface. When FlowLink is not in the same subnet as the self-IP, you must know the default gateway IP of the self-IP interface.

Create a Pool for Flow Collector

To create a pool of flow collectors to receive the flow record messages from the F5 system:

1. In the F5 UI, click **Main > Local Traffic > Pools > Pool Lists > Create**.
2. Enter a unique name in the **Name** field, which represents the flow collector.
3. A *Health Monitor* is not required. If you want to see if the F5 system can reach the flow collector, select **`gateway_icmp`** and move it to the Active box.
4. In the **New Member** section, configure the collector IP address.
5. Click **Add**.

   If you are using **IPFIX**, use the following configuration:

Field	Value
Node Name	Enter the Collector IP address
Service Port	4739

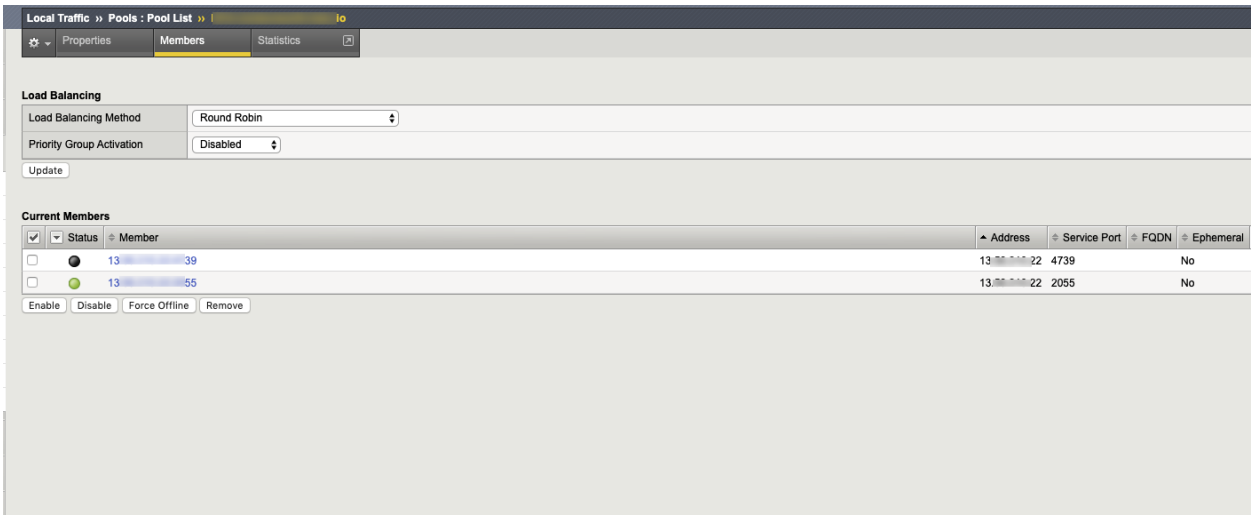
   If you are using **NetFlow**, use the following configuration:

Field	Value
Node Name	Enter the Collector IP address
Service Port	2055

6. Click **Finished**.

The below example shows two (2) different nodes configured in one pool. Both nodes have the IP address. However, one is for IPFIX and one is for NetFlow. Even though F5 allows two nodes in the pool, it is recommended to only have one node enabled (either 2055 or 4739).

Example with NetFlow enabled and IPFIX disabled:



Create a Log Destination

To create a log destination to stream the logs in either IPFIX or NetFlow V9 format to the Pool:

1. In the F5 UI, click **Main > System > Logs > Configuration > Log Destinations > Create**.
2. Enter a unique name in the **Name** field, which represents the flow collector.
3. In the **Type** field, select IPFIX.
4. Configure the IPFIX Settings.

   If you are using **IPFIX**, use the following configuration:

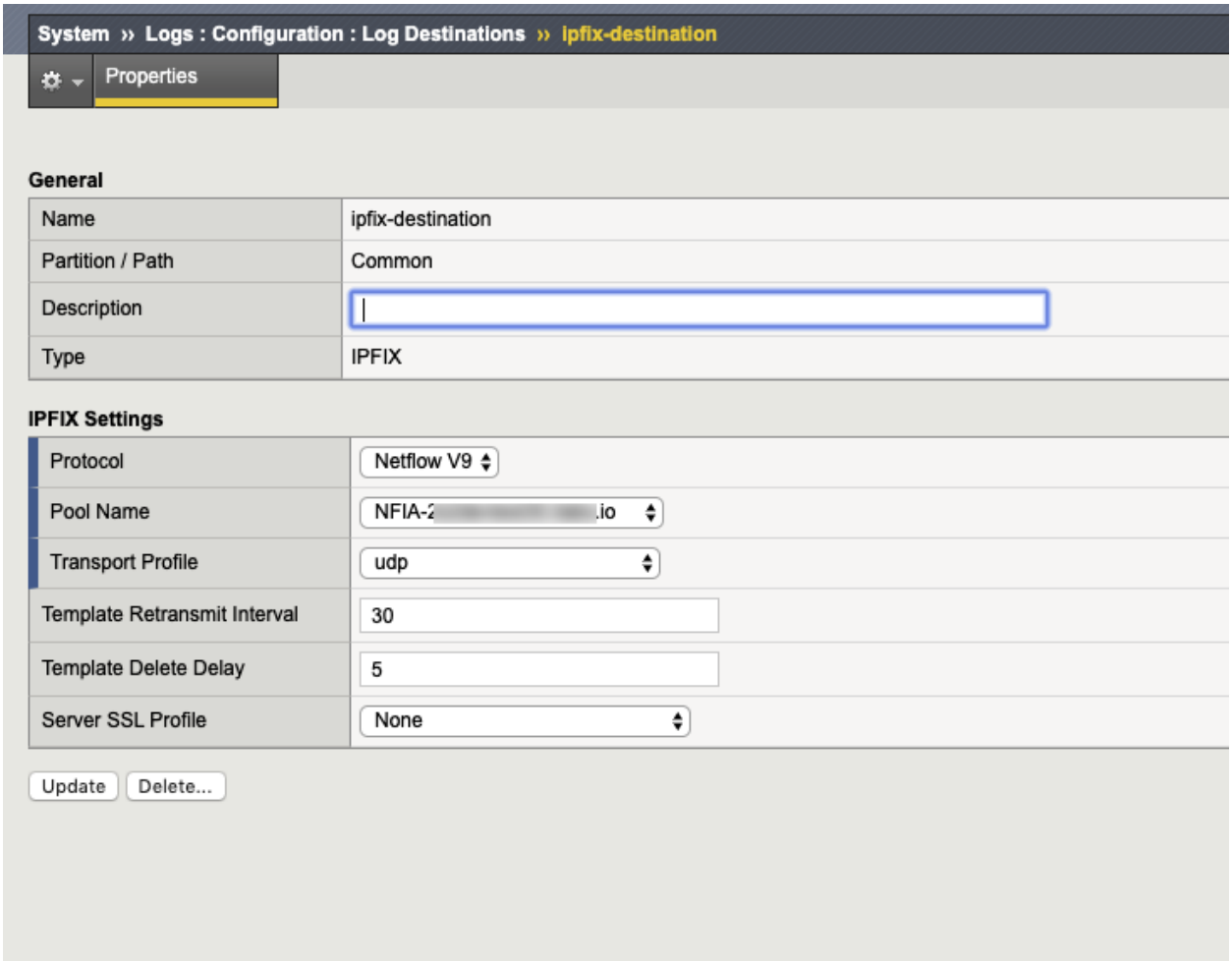
Field	Value
Protocol	Select IPFIX
Pool Name	Select the pool created earlier
Transport Profile	UDP

If you are using **NetFlow**, use the following configuration:

Field	Value
Protocol	Select NetFlow V9
Pool Name	Select the pool created earlier
Transport Profile	UDP

5. Click Finished.

Example of a Log Destination configuration with NetFlow:



Create a Log Publisher

To create a log publisher to send logs to the specified log destination:

1. In the F5 UI, click **Main > System > Logs > Configuration > Log Publishers > Create**.
2. Enter a unique name in the **Name** field, which represents the flow collector.
3. In the **Destination** field, move your log destination from *Available* to *Selected*.
4. Click **Finished**.

Create an iRule

To create an iRule to which it parses network traffic and sends flow records to the specified log publisher:

1. Go to **Main > iRules > iRule List > Create**.
2. Enter a unique name in the **Name** field, which represents the flow collector.

3. In the **Definition** text field, enter the rules for parsing traffic. Ensure the iRule points to the *log publisher* created earlier.
4. Click **Finished**.

In the iRule example shown below, replace <*insert_log_publisher_name_here*> with the name of the log publisher.

```
when RULE_INIT {
  set static::http_rule1_dest ""
  set static::http_rule1_tmplt ""
}


# CLIENT_ACCEPTED event to initiate IPFIX destination and template
when CLIENT_ACCEPTED {
  set start [clock clicks -milliseconds]
  if { $static::http_rule1_dest == ""} {
    # open the logging destination if it has not been opened yet
    set static::http_rule1_dest [IPFIX::destination open -publisher
/Common/<insert_log_publisher_name_here>]
  }
  if { $static::http_rule1_tmplt == ""} {
    # if the template has not been created yet, create the template
    set static::http_rule1_tmplt [IPFIX::template create "flowStartMilliseconds \
                                                          sourceIPv4Address \
                                                          sourceIPv6Address \
                                                          destinationIPv4Address \
                                                          destinationIPv6Address \
                                                          sourceTransportPort \
                                                          destinationTransportPort
\
                                                          protocolIdentifier \
                                                          octetTotalCount \
                                                          packetTotalCount \
                                                          octetDeltaCount \
                                                          packetDeltaCount \
                                                          postNATSourceIPv4Address
\
```

```
                                                        postNATSourceIPv6Address
\
                                                        postNATDestinationIPv4Ad
dress \
                                                        postNATDestinationIPv6Ad
dress \
                                                        postNAPTSourceTransportP
ort \
                                                        postNAPTDestinationTrans
portPort \
                                                        postOctetTotalCount \
                                                        postPacketTotalCount \
                                                        postOctetDeltaCount \
                                                        postPacketDeltaCount \
                                                        flowEndMilliseconds \ "]

  }
  set rule1_msg1 [IPFIX::msg create $static::http_rule1_tmplt]
}

# SERVER_CONNECTED event to initiate flow data to specified log publisher and
populate 5 tuples
when SERVER_CONNECTED {
  set client_closed_flag 0
  set server_closed_flag 0
  IPFIX::msg set $rule1_msg1 flowStartMilliseconds $start
  IPFIX::msg set $rule1_msg1 protocolIdentifier [IP::protocol]

  # Clientside
  if { [clientside {IP::version}] equals "4" } {
    # Client IPv4 address
    IPFIX::msg set $rule1_msg1 sourceIPv4Address [IP::client_addr]
    # BIG-IP IPv4 VIP address
    IPFIX::msg set $rule1_msg1 destinationIPv4Address [clientside {IP::local_
```

```
addr}]
  } else {
    # Client IPv6 address
    IPFIX::msg set $rule1_msg1 sourceIPv6Address [IP::client_addr]
    # BIG-IP IPv6 VIP address
    IPFIX::msg set $rule1_msg1 destinationIPv6Address [clientside {IP::local_
addr}]
  }
  # Client port
  IPFIX::msg set $rule1_msg1 sourceTransportPort [TCP::client_port]
  # BIG-IP VIP port
  IPFIX::msg set $rule1_msg1 destinationTransportPort [clientside {TCP::local_
port}]

  # Serverside
  if { [serverside {IP::version}] equals "4" } {
    # BIG-IP IPv4 self IP address
    IPFIX::msg set $rule1_msg1 postNATSourceIPv4Address [IP::local_addr]
    # Server IPv4 IP address
    IPFIX::msg set $rule1_msg1 postNATDestinationIPv4Address [IP::server_addr]
  } else {
    # BIG-IP IPv6 self IP address
    IPFIX::msg set $rule1_msg1 postNATSourceIPv6Address [IP::local_addr]
    # Server IPv6 IP address
    IPFIX::msg set $rule1_msg1 postNATDestinationIPv6Address [IP::server_addr]
  }
  # BIG-IP self IP port
  IPFIX::msg set $rule1_msg1 postNAPTSourceTransportPort [TCP::local_port]
  # Server port
  IPFIX::msg set $rule1_msg1 postNAPTDestinationTransportPort [TCP::server_port]
}

# SERVER_CLOSED event to collect IP pkts and bytes count on serverside
when SERVER_CLOSED {
  set server_closed_flag 1
  # when flow is completed, BIG-IP to server REQUEST pkts and bytes count
  IPFIX::msg set $rule1_msg1 octetTotalCount [IP::stats bytes out]
  IPFIX::msg set $rule1_msg1 packetTotalCount [IP::stats pkts out]
```

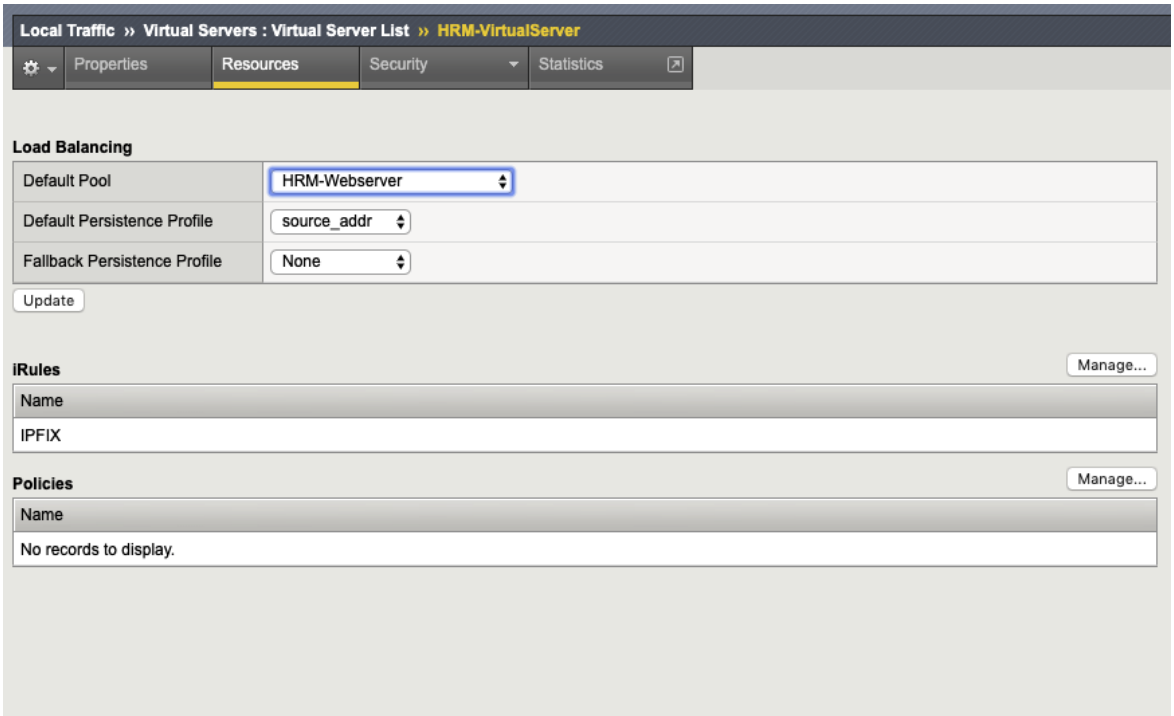
```
  # when flow is completed, server to BIG-IP RESPONSE pkts and bytes count
  IPFIX::msg set $rule1_msg1 octetDeltaCount [IP::stats bytes in]
  IPFIX::msg set $rule1_msg1 packetDeltaCount [IP::stats pkts in]
    IPFIX::destination send $static::http_rule1_dest $rule1_msg1
}

# CLIENT_CLOSED event to collect IP pkts and bytes count on clientside
when CLIENT_CLOSED {
  set client_closed_flag 1
  # when flow is completed, client to BIG-IP REQUEST pkts and bytes
octetDeltaCount
  IPFIX::msg set $rule1_msg1 postOctetTotalCount [IP::stats bytes in]
  IPFIX::msg set $rule1_msg1 postPacketTotalCount [IP::stats pkts in]
  # when flow is completed, BIG-IP to client RESPONSE pkts and bytes count
  IPFIX::msg set $rule1_msg1 postOctetDeltaCount [IP::stats bytes out]
  IPFIX::msg set $rule1_msg1 postPacketDeltaCount [IP::stats pkts out]
  # record the client closed time in ms
  IPFIX::msg set $rule1_msg1 flowEndMilliseconds [clock click -milliseconds]
    # send the IPFIX log
    IPFIX::destination send $static::http_rule1_dest $rule1_msg1
}
```

Apply the iRule to a Virtual Server

To apply the iRule to a virtual server whose traffic you want to parse:

1. Go to **Main > Virtual Server > Virtual Server List**.
2. Select the virtual server you want to monitor.
3. Click the **Resources** tab. In the iRule section, click **Manage**.
4. Select the **iRule** that you previously created and move the iRule from *Available* to *Enable*.
5. Click **Finished**.

   Example of a Virtual Server Resources page with the new iRule applied:



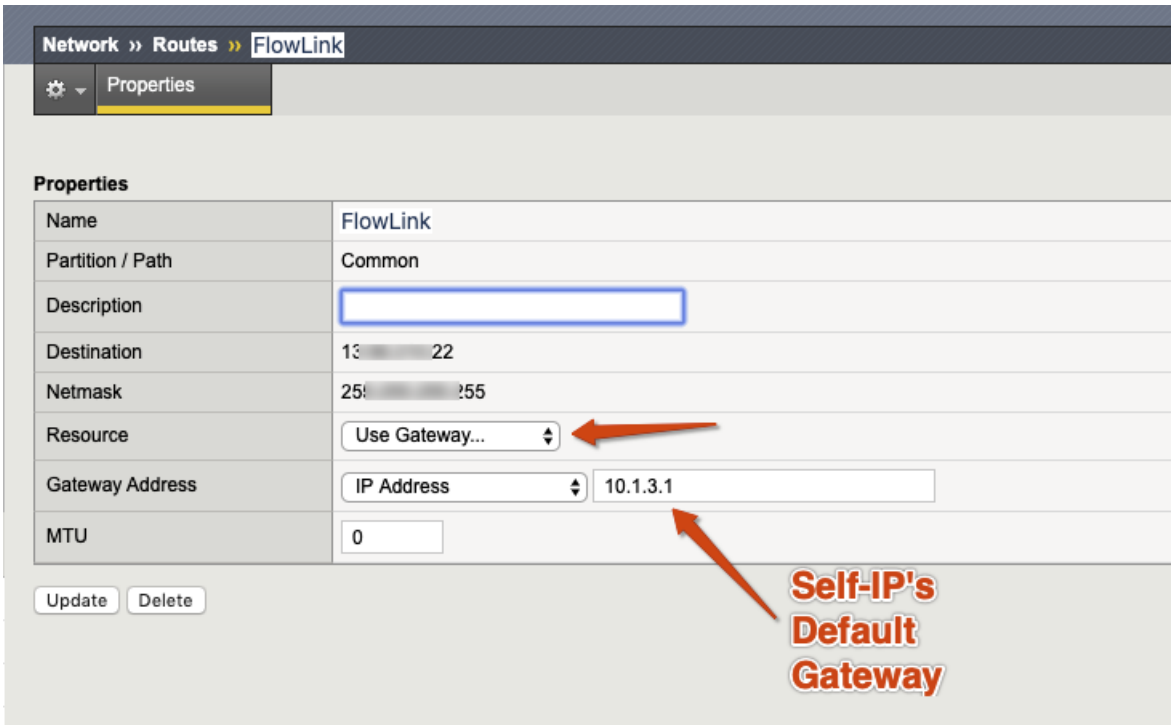
Create a Route Entry

By default, all traffic is sent out of the management interface. However, F5 does not support flow exports via the management NIC. You must add a route to force traffic, which is destined to the flow collector to leave a self-IP interface.

To create a route entry, if the F5 self-IP is unable to reach the flow collector:

1. In the F5 UI, click **Main > Network > Routes > Add**.
2. In the **Properties** section, create a route entry to send the flow records from F5 to the external flow collector IP address.

   For Resource, select the *Use Gateway* option.


Network » Routes » FlowLink
Properties
Properties
Name
FlowLink
Partition / Path
Common
Description
Destination
22
Netmask
255
Resource
Use Gateway...
Gateway Address
IP Address
10.1.3.1
MTU
0
Update
Delete
Self-IP's
Default
Gateway

Chapter 4

Troubleshooting

This section describes how to troubleshoot some issues when configuring or using FlowLink.

FlowLink not Receiving Data

1. Make sure iptables is turned *Off* on FlowLink, or make sure iptables is not blocking the ports that FlowLink is listening on.
2. Use `netstat -a` to make sure FlowLink is listening on the correct ports.

NOTE:
netstat has a bug, which shows that applications are only listening with IPv6 on listed ports, when they are actually listening on those ports with IPv4.

Unable to Ping or TCPdump on the F5 Self-IP Interface

1. SSH to F5 as an administrator.
2. List the interfaces to see the interface names.

```
admin@(ip-10-1-1-197)(cfg-sync Standalone)(Active)(/Common)(tmos)# show net
interface

-----------------------------------------------------------------
```

```
Net::Interface
Name  Status    Bits    Bits    Pkts    Pkts  Drops  Errs  Media
                  In     Out      In     Out
-------------------------------------------------------------------
1.1       up    1.3G    1.1G    2.6M    2.6M      0     0   none
1.2       up  177.7M  301.4M  298.9K  310.4K      0     0   none
mgmt      up  310.9G  876.6G  298.8M  325.5M      0     0   none
```

3. Run TCPdump to listen for traffic between Self-IP interface and flow collector IP.
4. Generate traffic while the TCPdump is running by either opening another SSH session and doing PING test or by sending normal traffic through the virtual server. If you turned on **health monitoring** with gateway_icmp enabled from the Create a Pool for Flow Collector section, then F5 should already generate ICMP traffic.

   The example shown below uses interface name *1.2* with flow collector *IP 13.56.210.22*. Health monitoring with *gateway_icmp* is enabled.

```
admin@(ip-10-1-1-197)(cfg-sync Standalone)(Active)(/Common)(tmos)# tcpdump -
ni 1.2 host 13.56.210.22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on 1.2, link-type EN10MB (Ethernet), capture size 65535 bytes
09:08:47.855318 IP 10.1.3.223 > 13.56.210.22: ICMP echo request, id 54351,
seq 37906, length 20 out slot1/tmm3 lis=
09:08:47.857694 IP 13.56.210.22 > 10.1.3.223: ICMP echo reply, id 54351, seq
37906, length 20 in slot1/tmm3 lis=
09:08:52.864852 IP 10.1.3.223 > 13.56.210.22: ICMP echo request, id 54354,
seq 37906, length 20 out slot1/tmm2 lis=
09:08:52.867091 IP 13.56.210.22 > 10.1.3.223: ICMP echo reply, id 54354, seq
37906, length 20 in slot1/tmm2 lis=
```

Network Connectivity

The flow to test network connectivity is:

- Network device > FlowLink
- FlowLink > PCE

TCPdump

To use TCPdump:

- Run on a network device to verify flow records are sent out.
- Run on FlowLink to verify flow records are coming in.

Debug Option

FlowLink has a debug option that displays:

- Incoming flow records
- IP, port, and protocol recorded for flow records
- Each time flows are aggregated and uploaded to the PCE
- PCE response code to `POST`

To debug FlowLink in the session, add the `--debug` flag to your FlowLink command.

Example with the debug option enabled:

```
CONFIG_FILE=/home/employee/config.yaml.netflow /usr/local/bin/illumio/flowlink --debug
```

IMPORTANT:
Using the debug flag, generates a large amount of data to the console. Enable this option only if needed.