OpenNTI Documentation

Release 1.0

Damien Garros, Efrain Gonzalez, Michael Pergament, Pablo Sagre

August 05, 2016

Contents

OpenNTI is a container packaged with all tools needed to collect and visualize time series data from network devices. Data can be collected from different sources:

- **Data Collection Agent** : Collect data on devices using CLI/Shell or Netconf
- **Data Streaming Collector** : Take all data streamed by Juniper devices as Input (JTI, Analyticsd, soon Openconfig with gRPC)
- **Statsd interface** : Accept any Statsd packets

It's pre-configured with all tools and with a default dashboard .. **Send it data, it will graph it**

Thanks to docker, it can run pretty much anywhere on server, on laptop ... on the device itself

More detailed description of a project can be found [here](http://forums.juniper.net/t5/Analytics/Open-Source-Universal-Telemetry-Collector-for-Junos/ba-p/288677) (including a series of videos on how to use it):

CHAPTER 1

Customize OpenNTI

1.1 Customize container's name and ports

All port numbers and names used by start/stop scripts are centralized in one file : open-nti.params, you can easily adapt this file with your own port numbers or names. It's mandatory if you are planning to run multiple instances of OpenNTI on the same server.

1.2 Customize the container itself

If you want to make some modifications, you can always build the container yourself using the script `` `./docker.build.sh` ``. >The first time you run "./docker.build.sh", it will take 10-15min to download and compile everything but after that it will be very fast

CHAPTER 2

How to report feedback / participate in the project

For any issues please open an [issue on Github](https://github.com/Juniper/open-nti/issues). For comments, suggestions or questions please use our [google group](https://groups.google.com/forum/#!forum/open-nti)

To participate, please: - Fork the project - Send us a pull request

> if you are planning significant changes, please start a discussion first.

Contributions are more than Welcome

2.1 How to install or upgrade

2.1.1 Requirements

The requirements is to have docker and docker-compose installed on your Linux server/machine. Instructions to install are available below - docker: http://docs.docker.com/engine/installation/ubuntulinux/ - docker-compose: https://docs.docker.com/compose/install/

It's also available for: - Mac: https://docs.docker.com/engine/installation/mac/ - Windows: https://docs.docker.com/engine/installation/windows/

2.1.2 How to Install/Start

OpenNTI is available on .. _[Docker Cloud]: https://hub.docker.com/r/juniper/open-nti/ and this project provide scripts to easily download/start/stop it.

```
git clone https://github.com/Juniper/open-nti.git
cd open-nti
./docker.start.sh
```

Note: On Ubuntu, you'll have to add "sudo" before the last command

By default it will start 3 containers and it's working in **non-persistent mode**, once you stop it all data are gone. It's possible to start the main container in **persistent mode** to save the database outside the container, b y using the startup script `docker.start.persistent.sh`. *Persistent mode on Mac OS requires at least v1.12*

2.1.3 How to update

It's recommended to upgrade the project periodically, both the files from github.com and the containers from Docker Hub. You can update easily with

```
./docker.update.sh
```

2.2 Architecture description [WIP]

OpenNTI architecture is designed to be modular. the main components are a Timeserie Database(influxdb) and a graphical interface (grafana)

Based on the need, containers can be added or removed to add functionalities.

2.2.1 Docker compose

All containers are started using docker-compose.yaml

./docker.start.sh

You can create your own docker-compose file and pass it

./docker.start.sh <my docker compose file>

2.2.2 List of available Plugins

JTI

Event / Syslog

Input plugin container

- https://github.com/Juniper/open-nti-input-syslog
- https://github.com/Juniper/open-nti-input-jti

2.3 Data Collection Agent

2.3.1 Configuration

data/hosts.yaml In data/hosts.yaml you need to provide the list of devices you want to pull information from For each device, you need to indicate the name ane one or multiple *tags* (at least one). Tags will be used later to know which credentials should be used for this device and which commands need to be executed

```
<hostA>: <tag1> <tag4>
<hostB>: <tag1> <tag4>
<hostC>: <tag2> <tag4> <tag5>
<hostD>: <tag1> <tag4>  <--- Those tags relate the Hosts with the credentials and the commands to use
```

Example

```
mx-edge011: edge mx madrid bgp mpls
mx-agg011: agg mx madrid bgp isis
qfx-agg022: agg qfx munich bgp
qfx5100-02: tor qfx madrid isis
```

Note: The default configuration assume that hosts defined in hosts.yaml can be resolved with DNS if your hosts doesn't have DNS entry, it's possible to indicate the IP address in the hosts.yaml file instead of the name

192.168.0.1: edge mx madrid bgp mpls

To avoid using Ip addresses in the dashboard, you can use the device hostname defined in the configuration instead of the value define in hosts.yaml by setting the parameter **use_hostname** to true in **open-nti.variables.yaml**
use_hostname: True

data/credentials.yaml

You need to provide at least one credential profile for your devices

```
jdi_lab:
  username: '*login*'          (Single quote is to force to be imported as string)
  password: '*password*'       (Single quote is to force to be imported as string)
  method: password             (other supported methods 'key' and 'enc_key' for ssh Key-Based Authenti
  key_file: ./data/*key_file*  (optional: only appies if method key or enc_key is used, it must be loc
  tags: tag1 tag2
```

data/commands.yaml

```
generic_commands:  <--- You can name the group as best fits you
   commands: |
      show version | display xml  <--- There is no limit on how many commands can be added into a gro
      show isis statistics | display xml <-- Before adding a command, confirm that there is a relate
      show system buffers
      show system statistics icmp | display xml
      show route summary | display xml
   tags: tag1 tag2
```

2.3.2 Execution periodic

To collect data periodically with the **Data Collection Agent**, you need to setup a cron job inside the container. As part of the project, open-nti is providing some scripts to easily add/remove cron jobs **inside** the container **from** the host.

Scripts provided:

- **open-nti-start-cron.sh**: Create a new cron job inside the container
- **open-nti-show-cron.sh**: Show all cron jobs configured inside the container
- **open-nti-stop-cron.sh**: Delete a cron job inside the container for a specific tag

To start cron job to execute commands specified above for specific tag every minute:

```
./open-nti-start-cron.sh 1m '--tag tag1'
```

To start cron job for more than one tag at the same time:

```
./open-nti-start-cron.sh 1m '--tag tag1 tag2'
```

To start cron job to execute commands specified above for specific tag every 5 minutes:

```
./open-nti-start-cron.sh 5m '--tag tag1'
```

To start cron job to execute commands specified above for specific tag every hour:

```
./open-nti-start-cron.sh 1h '--tag tag1'
```

To show all scheduled cron jobs:

```
./open-nti-show-cron.sh 'all'
```

To stop cron job for specific tag:

```
./open-nti-stop-cron.sh '--tag tag1'
```

Note: If you want to configure the cron job yourself, open-nti use this command: `/usr/bin/python /opt/open-nti/open-nti.py -s --tag <tag>`

2.4 Data Streaming Collector

Currently the collector accept: - Analyticsd (qfx5k) streams in JSON/UDP on port **UDP/50020** - Juniper Telemetry Interface (MX/PTX) streams in GPB/UDP on port **UDP/50000**

Important: **it's important that all devices have the correct time defined**, it's recommended to configure NTP everywhere

2.4.1 statsd interface

open-nti is using telegraf to support statsd Statsd is a popular tool to send metrics over the network, it has been designed by etsy More information below : - https://github.com/etsy/statsd/blob/master/docs/metric_types.md - https://github.com/influxdata/telegraf/tree/master/plugins/inputs/statsd

Here is an example of how to insert statsd data into the Database

```
root@d3e82264a08b:/# echo "opennti,device=qfx5100,type=int.rx:100|g" | nc -w 1 -u 127.0.0.1 8125
```

opennti define the serie device=qfx5100,type=int.rx will be converted as tag1 100 is the value g indicate gauge

2.5 Events

By default dashboards are configured to display some "events" that are stored in the database into the serie "events" Their are multiple ways to record entry in the events serie

2.5.1 Insert events via syslog

open-nti will access events in the syslog format on port **UDP/6000**. The goal is not to send all syslog but only relevant information like Commit or Protocol Flaps

To send only one syslog at commit time you can use the configuration below

```
set system syslog host 192.168.99.100 any any
set system syslog host 192.168.99.100 match UI_COMMIT_COMPLETED
set system syslog host 192.168.99.100 port 6000
```

2.5.2 Insert events in the database directly

It's possible to insert events with just a HTTP POST request to the database, here is an example using curl

```
curl -i -XPOST 'http://10.92.71.225:8086/write?db=juniper' --data-binary 'events,type=Error text="BGP
curl -i -XPOST 'http://10.92.71.225:8086/write?db=juniper' --data-binary 'events,device=qfx5100-01,ty
```

Note: any system that knows how to generate a HTTP POST request can inject an event. its very utile if you have a script/tool that run some tests to keep track of when major events happen

2.6 Dashboard Generator [WIP]

I created a Dashboard generator based on Python and Jinja2. It's been an open item for long time and it was too often in my way so I decided to take a stab at it. It's still very early stage and I'm sharing it to get feedback as early as possible.

Top of my head, I can think of multiple tasks for which it will help:

Convert JTI graphs to the new variables names Create graphs for the new JTI sensors (LSP, FW etc ..) Add templating for interface Create Dashboard for Netconf in mode 2 & 3 Create Dashboard on demand and more personalized In a nutshell, I templatized a grafana dashboard into multiple pieces:

The skeleton of the dashboard - The rows, composed of multiple panels or graphs - The graphs - The annotations - The templatings

To generate a dashboard you need to create a yaml file that indicate: the title, which rows, which annotations etc ..

```
title: Data Steaming Collector ALPHA
template: "dashboard_base.j2"

tags:
  - opennti

rows:
  - int-traffic.yaml
  - int-queue.yaml
  - int-buffer.yaml

templatings:
  - host_regex.yaml
  - interface.yaml
```

To generate the dashboard based on this config file, you just have to call this command line

```
cd dashboards/
python gendashboard.py --file data_streaming_collector.yaml
```

The rows are defined in the directory *templates/rows/* and the graphs in the directory *templates/graphs/* The idea is to define which template for each configuration file, so we don't need to turn everything into a variable in the templates. If 2 graphs are very different we can just have different templates.

It will keep the YAML file light and easily readable

2.7 Troubleshooting Guide

To check if containers are running, execute the following command. By default you should have 3 containers running .. code-block:: text

    docker ps

To force containers to stop, execute .. code-block:: text

    ./docker.stop.sh

To access the CLI of the main container for debug, Start a SSH session using the insecure_key provided in the repo and the script “docker.cli.sh” .. code-block:: text

    chmod 600 insecure_key ./docker.cli.sh

For the Input containers named __open-nti-input-*__ you can access the logs directly from docker by running : .. code-block:: text

    docker logs <container name or ID>

2.7.1 FAQ

I’m streaming data from devices but I’m not seeing anything on the Dashboard

To reach the dashboard, traffic have to go through the following path: **Device** >(A)> **Host** >(B)> **Container** >(C)> **Fluentd** >(B)> **InfluxDB** >(E)> **Grafana**

A - Check that traffic is reaching the Host

The best solution is to use TCPDUMP on the Host and filter on destination port .. code-block:: text

    On Unix/Mac tcpdump -i <ingress interface> -n dst port <dest port number>

B - Check that traffic is reaching the container The best solution is to use TCPDUMP inside the container .. code-block:: text

    ./docker.cli.sh tcpdump -i eth0 -n dst port <dest port number>

    RPF check might be a problem if you see incoming packets in A but not in B. If you e.g. use Src IP for which there is no route entry on host OS (Ubuntu does RPF check as default), packets would be discarded.

C - Check Fluentd Check fluentd logs, inside the container .. code-block:: text

    ./docker.cli.sh tail -f /var/log/fluentd.log

Nothing should be printed if everything is right

D - Check if data is properly reaching the database - connect on Influxdb management interface with a browser on port 8083 - Select Juniper as database on top right corner - Run query `show measurements` to see what is present - Execute query for `SELECT * FROM "<measurements>"`

> Destination tables will vary depending of the incoming traffic > - For MX > jnpr.jvision > - For QFX5100/EX4300 > jnpr.analyticsd

CHAPTER 3

Indices and tables

- genindex
- modindex
- search