

Тестируем Junos Telemetry Interface (JTI) в связке с OpenNTI

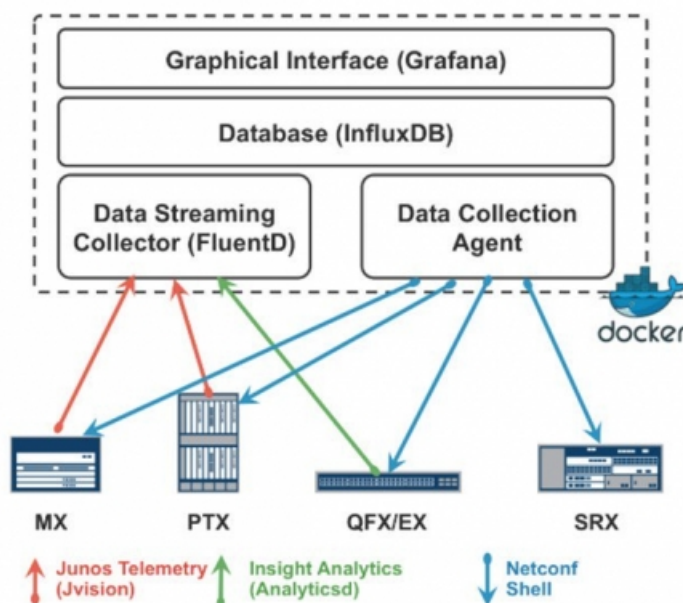
В сети на данный момент достаточно публикаций на английском языке с описанием и анализом обеих технологий.

Эта статья не является переводом какой-либо из них и основана на собственном опыте тестирования OpenNTI и JTI. Оба продукта представлены компанией Juniper и являются частью SDN-решений в плане визуализации различных метрик, получаемых с сетевых устройств.

Начнем с описания OpenNTI, поддерживаемого Juniper проектом с открытым исходным кодом.

OpenNTI — упакованный в docker (единый контейнер) набор инструментов для сбора, хранения и визуализации данных, полученных от устройств сети. В его состав входят популярный графический интерфейс Grafana, предназначенный для работы со streaming data, база данных InfluxDB, коллектор данных FluentD и агент PyEZ — библиотека для Python от Juniper, разработанный для удаленного управления и автоматизации сетевых устройств. Для сбора данных по SNMP имеется SNMP агент.

Структура контейнера и взаимодействие с сетевыми устройствами отражены ниже.



Для OpenNTI характерны push и pull методы сбора данных телеметрии с устройств сети. Push — данные о телеметрии отправляются настраиваемыми сенсорами устройств в виде потока данных по предустановленным таймерам, pull — данные с устройств запрашиваются, а затем обрабатываются агентом. В отношении push применимы два формата передачи данных: Native Streaming и OpenConfig Streaming. Оба используют формат данных gpb (Google protocol buffers) и отличаются тем, что Native Streaming использует в качестве транспорта UDP и отправляет данные с line card, а OpenConfig Streaming передает данные от Routing Engine через gPRC over HTTP2.

С проектом более подробно можно ознакомиться по ссылкам:

- [Juniper/open-nti](#),
- [OpenNTI Documentation](#),
- [Continuous Integration for OpenNTI Collector without any Junos Device](#).

В последнее время популярной моделью сбора, хранения и обработки метрик сетевых устройств становится асинхронная push модель, имеющая в отличие от традиционной pull модели, используемой в SNMP и CLI, более высокую масштабируемость и эффективность.

Одним из вендорских решений является Junos Telemetry Interface (JTI), представленный в JunOS начиная с 15.1F3 для MX-серии, и в более поздних релизах для других продуктовых линеек. Технология, по сути, довольно проста и предполагает сбор и передачу данных о системных ресурсах (физические интерфейсы, firewall-фильтры, etc.) с предварительно сконфигурированных сенсоров. Поддерживаются Native Streaming и OpenConfig Streaming.

Более подробно описано по ссылкам ниже:

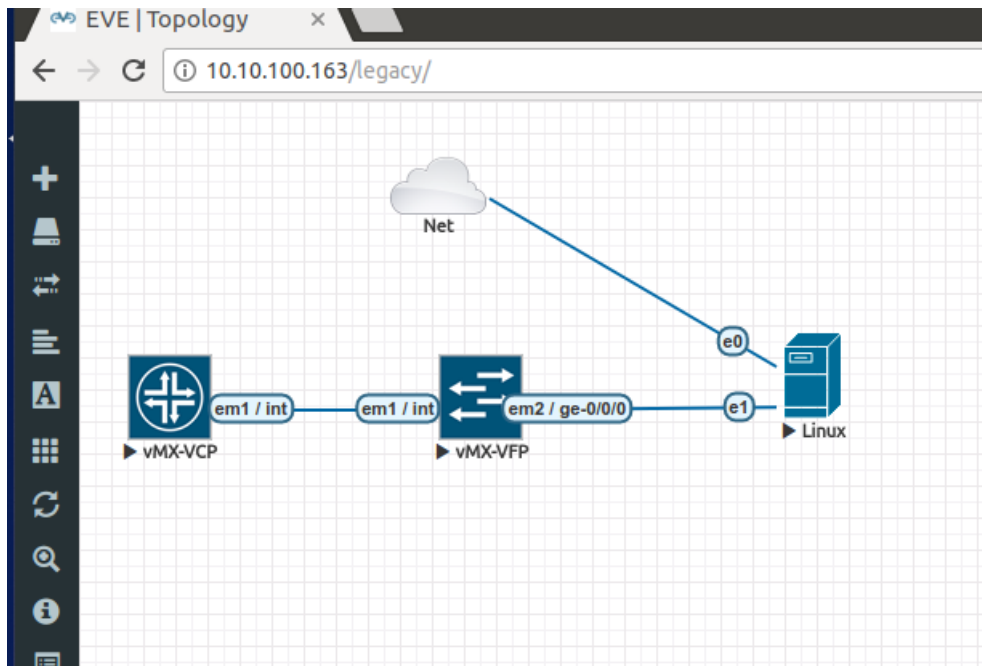
- [Overview of the Junos Telemetry Interface](#),
- [Junos Telemetry Interface Feature Guide](#),
- [Configuring a Junos Telemetry Interface Sensor \(CLI Procedure\)](#),

[Все описанное выше](#) можно считать небольшим введением в обе технологии, практические аспекты применения которых будут рассмотрены ниже в примерах.

Итак, прежде чем внедрять данные решения в "живую" сетевую инфраструктуру, было принято решение развернуть и протестировать их на лабораторном стенде.

В качестве стенда использовался сетевой эмулятор EVE-NG (старое название Unetlab, <http://www.eve-ng.net/>) с установленными Juniper vMX и виртуальным хостом Ubuntu Server 16.04.3, предназначенным для OpenNTI. Описание развертывания эмулятора и установка хостов выходит за рамки данной статьи. Единственное, с чем пришлось столкнуться, это с правами для виртуальных сетевых адаптеров VMware Player, без которых для хоста с Ubuntu отсутствовал доступ в Internet. Проблема решилась следующим образом, на физической машине изменены права для трех адаптеров VMware:

```
chmod q+rw /dev/vmnet0  
chmod q+rw /dev/vmnet1  
chmod q+rw /dev/vmnet8
```



Установка и запуск OpenNTI на виртуальный хост из docker довольно проста и выполняется в несколько кликов.

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

В процессе запуска могут возникнуть ошибки со ссылкой на отсутствующие пакеты make и docker-compose, установить которые можно с помощью:

```
apt-get install make  
apt-get install docker-compose
```

Существует возможность изменения дефолтных параметров контейнера.

```
root@ubuntu-noc2:~/open-nti# nano open-nti.params
```

```
GNU nano 2.5.3                                     файл: open-nti
! /bin/bash

## Image and container
export MAIN_IMAGE_NAME=juniper/open-nti
export MAIN_CONTAINER_NAME=opennti_con

export INPUT_JTI_DIR=plugins/input-jti
export INPUT_JTI_IMAGE_NAME=juniper/open-nti-input-jti
export INPUT_JTI_CONTAINER_NAME=opennti_input_jti

export INPUT_SYSLOG_DIR=plugins/input-syslog
export INPUT_SYSLOG_IMAGE_NAME=juniper/open-nti-input-syslog
export INPUT_SYSLOG_CONTAINER_NAME=opennti_input_syslog

export INPUT_NETCONF_DIR=plugins/input-netconf
export INPUT_NETCONF_IMAGE_NAME=juniper/open-nti-input-netconf
export INPUT_NETCONF_CONTAINER_NAME=opennti_input_netconf

export INPUT_OC_DIR=plugins/input-oc
export INPUT_OC_IMAGE_NAME=juniper/open-nti-input-oc
export INPUT_OC_CONTAINER_NAME=opennti_input_oc

export INPUT_SNMP_DIR=plugins/input-snmp
export INPUT_SNMP_IMAGE_NAME=juniper/open-nti-input-snmp
export INPUT_SNMP_CONTAINER_NAME=opennti_input_snmp

export INPUT_INTERNAL_DIR=plugins/input-internal
export INPUT_INTERNAL_IMAGE_NAME=juniper/open-nti-input-internal
export INPUT_INTERNAL_CONTAINER_NAME=opennti_input_internal

export LB_UDP_DIR=plugins/lb-udp
export LB_UDP_IMAGE_NAME=juniper/open-nti-lb-udp
export LB_UDP_CONTAINER_NAME=opennti_lb_udp

export MAIN_CONTAINER_NAME=opennti_con

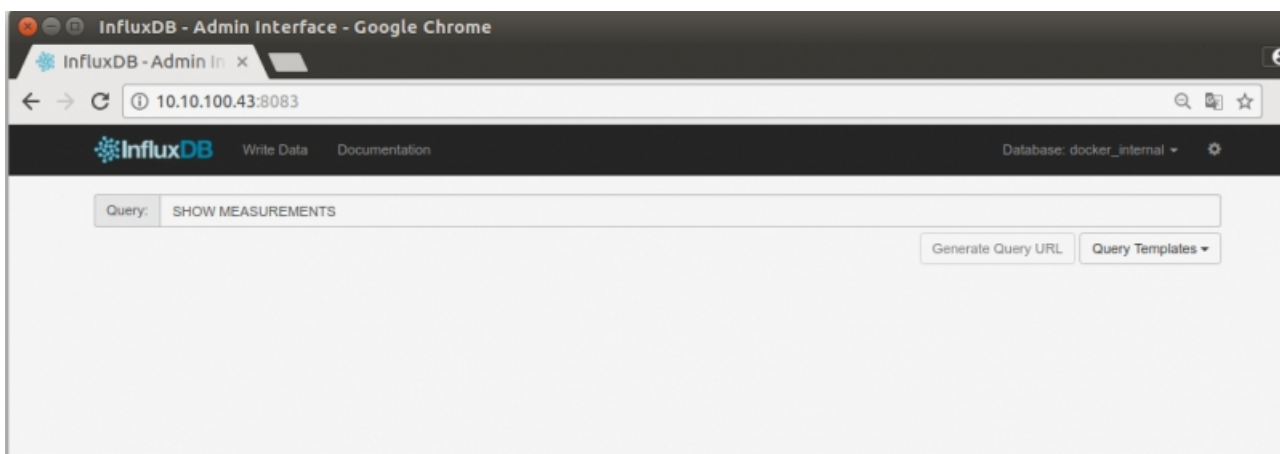
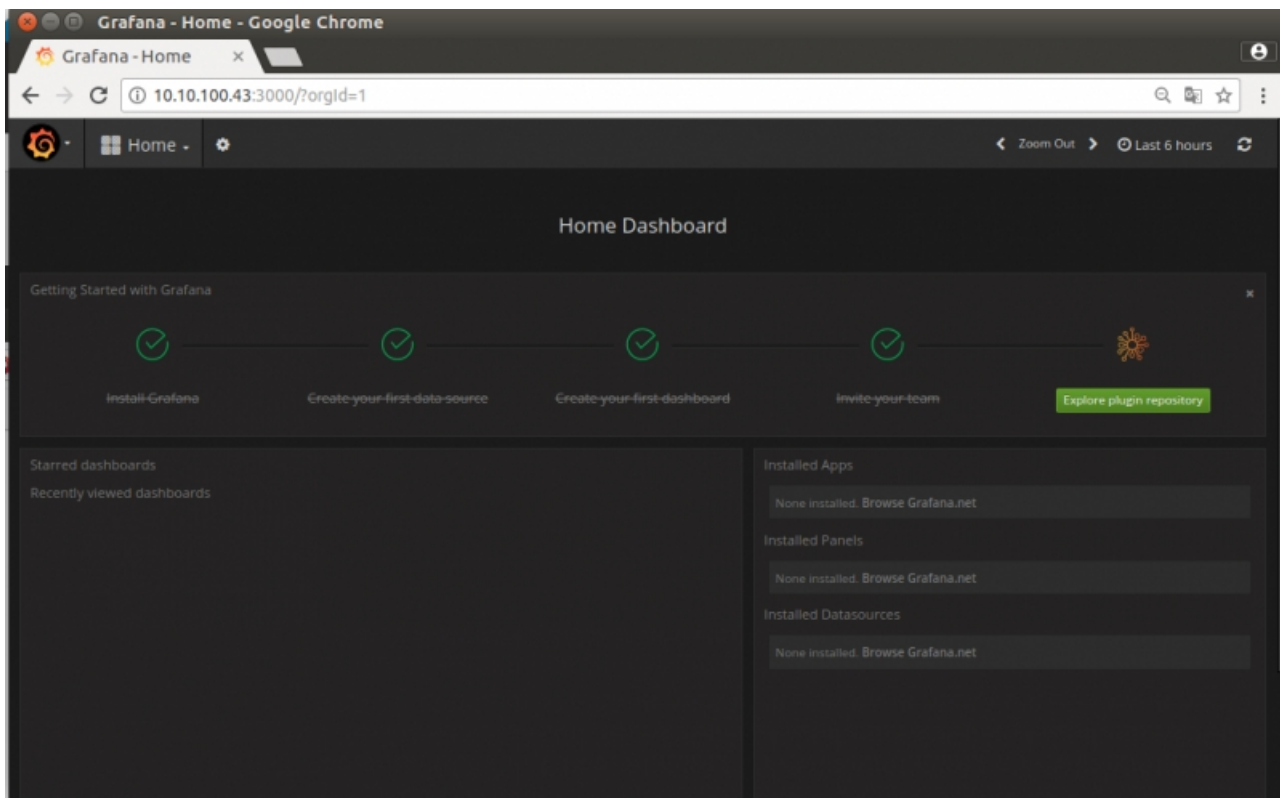
export DOCKER_COMPOSE_FILE=docker-compose.yml

## Local ports that will be redirected to the Open NTI
## Startup will fail if some ports are already in use
export LOCAL_PORT_JTI=50000
export LOCAL_PORT_NA=50010
export LOCAL_PORT_ANALYTICSD=50020
export LOCAL_PORT_STATSD=8125
export LOCAL_PORT_SNMP=162
export LOCAL_PORT_EVENT=6000
export LOCAL_PORT_GRAFANA=3000
export LOCAL_PORT_INFLUXDB=8083
export LOCAL_PORT_INFLUXDB_API=8086
export LOCAL_PORT_NGINX=80
export LOCAL_PORT_SSH=22222

## Local directories that will be mapped into the container
export LOCAL_DIR_DATA=data
export LOCAL_DIR_LOG=logs
export LOCAL_DIR_DB=db
export LOCAL_DIR_DB_DATA=${LOCAL_DIR_DB}/data
export LOCAL_DIR_DB_VAR=${LOCAL_DIR_DB}/influxdb
export LOCAL_DIR_TESTS=tests
export LOCAL_DIR_DASHBOARD=dashboards

^G Помощь      ^O Записать    ^W Поиск      ^K Вырезать    ^J Выровнять   ^C
^X Выход      ^R ЧитФайл    ^M Замена     ^U Отмен. вырезку ^T Пров. синтак. ^_
```

После установки можно сразу же проверить доступ в Grafana и InfluxDB.



Для сбора данных методом Pull с сетевых устройств необходимо в каталоге `/open-nti/data/` отредактировать три файла:

`commands.yaml`
`credentials.yaml`
`hosts.yaml`

```
root@ubuntu-noc2:~/open-nti/data# ls -l
итого 24
-rw-r--r-- 1 root root 1061 ноя 20 13:28 commands.yaml
-rw-r--r-- 1 root root 105 ноя 20 13:27 credentials.yaml
-rw-r--r-- 1 root root 29 ноя 20 13:26 hosts.yaml
drwxr-xr-x 2 root root 4096 ноя 20 17:40 junos_parsers
-rw-r--r-- 1 root root 736 ноя 20 13:04 open-nti.variables.yaml
drwxr-xr-x 2 root root 4096 ноя 20 13:04 pfe_parsers
root@ubuntu-noc2:~/open-nti/data#
```

1)

```
root@ubuntu-noc2:~/open-nti/data# nano hosts.yaml
#test: test
192.168.5.1: lab
```

192.168.5.1 — IP-адрес интерфейса lo0 нашего маршрутизатора;
lab — тэг

Очень удобная раскраска по тэгам позволяет группировать устройства по классам, географическому положению, функционалу и собирать с них данные, обозначенные конкретными тэгами. Тэгов может

быть несколько на одно устройство, так же, как и одинаковые тэги могут применяться к различным устройствам.

2)

```
root@ubuntu-noc2:~/open-nti/data# nano credentials.yaml
lab_credentials:
  username: lab
  password: lab123
  method: password
  key_file:
  tags: lab
```

username=lab и tags=lab — не что иное как совпадение.

3)

```
root@ubuntu-noc2:~/open-nti/data# nano commands.yaml
# GENERIC COMMANDS
#NOTE: Commands will be executed & parsed as text/regex ONLY if there isn't any valid xml output for that
command
# (so please check the output of the command with "| display xml" before building the parser or adding it
in this file)
generic_commands:
  commands: |
    show isis statistics | display xml
    show chassis routing-engine | display xml
    show pfe statistics traffic | display xml
    show system processes extensive
    show system buffers
    show system statistics icmp | display xml
    show route summary | display xml
  tags: lab
lab_commands:
  commands: |
    show route summary | display xml
  tags: lab
# Commands to query vmx-docker-lwafr Container running vMX for lightweight 4 over 6.
# Note: The space in "statisti s" is required until PR-1236994 is fixed in Junos.
lwafr_commands:
  commands: |
    show lwafr statisti s | display xml
    show lwafr state | display xml
  tags: lab
# Do not remove this three dashes ("---") they are used to separate documents
---
```

Последний файл содержит список команд, по которым агент будет опрашивать устройства и получать данные в формате xml. Набор команд ограничивается содержимым в каталогах junos_parsers, но при желании каталог можно дополнить собственными парсерами. Но даже сейчас их представлено довольно прилично:

```
root@ubuntu-noc2:~/open-nti/data/junos_parsers# ls -l
итого 236
-rw-r--r-- 1 root root 276 ноя 20 13:04 file-list-detail-var-rundb-juniper.eph_iceaaa.parser.yaml
-rw-r--r-- 1 root root 2878 ноя 20 13:04 rtsockmon.parser.yaml
-rwxr-xr-x 1 root root 943 ноя 20 13:04 show-bfd-session-summary.parser.yaml
-rw-r--r-- 1 root root 709 ноя 20 13:04 show-bgp-neighbor-10.255.0.206.parser.yaml
-rw-r--r-- 1 root root 4389 ноя 20 13:04 show-bgp-summary.parser.yaml
-rw-r--r-- 1 root root 346 ноя 20 13:04 show-chassis-fpc.parser.yaml
-rw-r--r-- 1 root root 1437 ноя 20 13:04 show-chassis-routing-engine.parser.yaml
-rw-r--r-- 1 root root 1163 ноя 20 13:04 show-ddos-protection-protocols-statistics-terse.parser.yaml
-rw-r--r-- 1 root root 447 ноя 20 13:04 show-firewall.parser.yaml
-rw-r--r-- 1 root root 8539 ноя 20 13:04 show-igmp-snooping-statistics.parser.yaml
-rw-r--r-- 1 root root 968 ноя 20 13:04 show-interfaces-media.parser.yaml
-rw-r--r-- 1 root root 2835 ноя 20 13:04 show-isis-statistics.parser.yaml
-rw-r--r-- 1 root root 1621 ноя 20 13:04 show-krt-state.parser.yaml
-rw-r--r-- 1 root root 656 ноя 20 13:04 show-ldp-traffic-statistics.parser.yaml
```

```

-rw-r--r-- 1 root root 888 ноя 20 13:04 show-lwafr-state.parser.yaml
-rw-r--r-- 1 root root 5449 ноя 20 13:04 show-lwafr-statistics.parser.yaml
-rw-r--r-- 1 root root 580 ноя 20 13:04 show-mpls-lsp.parser.yaml
-rw-r--r-- 1 root root 3027 ноя 20 13:04 show-network-access-aaa-radius-servers-detail.parser.yaml
-rw-r--r-- 1 root root 1140 ноя 20 13:04 show-network-access-aaa-statistics-address-assignment-
pool.parser.yaml
-rw-r--r-- 1 root root 5513 ноя 20 13:04 show-pfe-statistics-traffic.parser.yaml
-rw-r--r-- 1 root root 1059 ноя 20 13:04 show-pppoe-statistics.parser.yaml
-rw-r--r-- 1 root root 4263 ноя 20 13:04 show-route-summary.parser.yaml
-rw-r--r-- 1 root root 935 ноя 20 13:04 show-route-summary.table.bgp.l3vpn.0.parser.yaml
-rw-r--r-- 1 root root 924 ноя 20 13:04 show-route-summary.table.inet.0.parser.yaml
-rw-r--r-- 1 root root 924 ноя 20 13:04 show-route-summary.table.inet.3.parser.yaml
-rw-r--r-- 1 root root 926 ноя 20 13:04 show-route-summary.table.inet6.0.parser.yaml
-rw-r--r-- 1 root root 934 ноя 20 13:04 show-route-summary.table.l2circuit.0.parser.yaml
-rw-r--r-- 1 root root 3298 ноя 20 13:04 show-security-ipsec-statistics.parser.yaml
-rw-r--r-- 1 root root 438 ноя 20 13:04 show-services-l2tp-summary.parser.yaml
-rw-r--r-- 1 root root 393 ноя 20 13:04 show-services-nat-pool-detail.parser.yaml
-rw-r--r-- 1 root root 644 ноя 20 13:04 show-services-rpm-probe-results.parser.yaml
-rw-r--r-- 1 root root 895 ноя 20 13:04 show-services-stateful-firewall-flow-analysis.parser.yaml
-rw-r--r-- 1 root root 918 ноя 20 13:04 show-services-stateful-firewall-subscriber-analysis.parser.yaml
-rw-r--r-- 1 root root 995 ноя 20 13:04 show-services-video-monitoring-mdi-flow-fpc-slot-1.parser.yaml
-rw-r--r-- 1 root root 379 ноя 20 13:04 show-snmp-get-jnxOperatingCPU-FPC.yaml
-rw-r--r-- 1 root root 378 ноя 20 13:04 show-snmp-get-jnxOperatingCPU-RE.yaml
-rw-r--r-- 1 root root 545 ноя 20 13:04 show-snmp-statistics.parser.yaml
-rw-r--r-- 1 root root 2068 ноя 20 13:04 show-subscribers-summary.parser.yaml
-rw-r--r-- 1 root root 253 ноя 20 13:04 show-subscribers-summary-port.parser.yaml
-rw-r--r-- 1 root root 1284 ноя 20 13:04 show-system-buffers.parser.yaml
-rw-r--r-- 1 root root 582 ноя 20 13:04 show-system-configuration-database-usage.parser.yaml
-rw-r--r-- 1 root root 251 ноя 20 13:04 show-system-core-dumps.parser.yaml
-rw-r--r-- 1 root root 5024 ноя 20 13:04 show-system-processes-extensive.parser.yaml
-rw-r--r-- 1 root root 5832 ноя 20 13:04 show-system-resource-monitor-summary.parser.yaml
-rw-r--r-- 1 root root 629 ноя 20 13:04 show-system-statistics-icmp.parser.yaml
-rw-r--r-- 1 root root 364 ноя 20 13:04 show-system-storage.parser.yaml
-rw-r--r-- 1 root root 756 ноя 20 13:04 show-system-virtual-memory.parser.yaml
-rw-r--r-- 1 root root 301 ноя 20 13:04 show-task-accounting.parser.yaml
-rw-r--r-- 1 root root 650 ноя 20 13:04 show-task-io.parser.yaml
-rw-r--r-- 1 root root 448 ноя 20 13:04 show-task-memory.parser.yaml
-rw-r--r-- 1 root root 357 ноя 20 13:04 show-version.parser.yaml

```

После редактирования запускаем работу агента по крону:

```
make cron-add TAG=lab TIME=1m
```

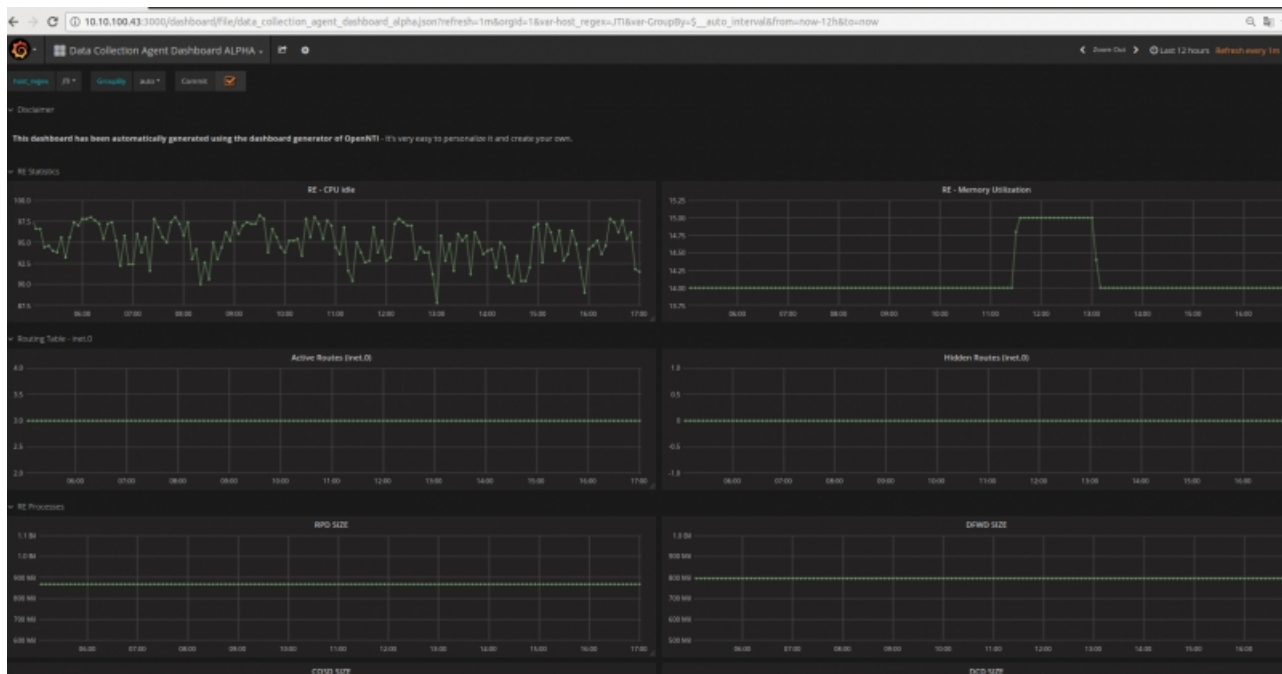
Самое важное, на маршрутизаторе обязательно необходимо включить ssh и netconf:

```

root@JTI# show system services
ssh;
netconf {
  ssh;
}
[edit]

```

Т.к. в Grafana templates предустановлены, можно сразу же увидеть в графике поступающие данные:



Обратите внимание на соответствующее название дашборда.

Для приема и обработки данных JTI каких-либо дополнительных конфигураций OpenNTI не требуется. Соответствующий плагин интегрирован в OpenNTI. Конфигурация выполняется исключительно на стороне маршрутизатора и выглядит вот так:

```
root@JTI# show services
analytics {
  traceoptions {
    file JTI size 2m files 5;
    flag all;
  }
  streaming-server NTI {
    remote-address 192.168.210.1;
    remote-port 50000;
  }
  export-profile export-jti {
    local-address 192.168.5.1;
    local-port 21111;
    dscp 20;
    forwarding-class assured-forwarding;
    reporting-rate 1;
    payload-size 3000;
    format gpb;
    transport udp;
  }
  sensor interface-1 {
    server-name NTI;
    export-name export-jti;
    resource /junos/system/linecard/interface/;
    resource-filter ge-*;
  }
  sensor interface-2 {
    server-name NTI;
    export-name export-jti;
    resource /junos/system/linecard/interface/logical/usage/;
  }
  sensor interface-3 {
    server-name NTI;
    export-name export-jti;
    resource /junos/system/linecard/cpu/memory/;
  }
}
[edit]
```

Первый блок traceoptions является отладочным, оставляем его без внимания. Блок streaming-server

указывает параметры сервера, куда будут экспортироваться данные. Таких серверов может быть несколько. 50000 — порт, на котором OpenNTI слушает JTI. Блок export-profile определяет такие параметры как IP-адрес и порт, с которых данные будут отправляться на коллектор, параметры CoS для этих данных, формат данных и транспорт о которых говорилось выше, частота отправки данных, в примере 1 сек, и payload-size. О последнем несколько слов в отдельности, так как при дефолтных настройках возникла проблема с размером пакетов, которая решилась подбором данного параметра, о чем показано ниже:

```
16:25:09.855373 IP 192.168.5.1.21111 > 192.168.210.1.3000: UDP, bad length 3240 > 1472
16:25:31.839582 IP 192.168.5.1.21111 > 192.168.210.1.3000: UDP, bad length 3239 > 1472
16:25:53.628087 IP 192.168.5.1.21111 > 192.168.210.1.3000: UDP, bad length 3240 > 1472
16:25:55.653492 IP 192.168.5.1.21111 > 192.168.210.1.3000: UDP, length 450
16:25:55.653575 IP 192.168.5.1.21111 > 192.168.210.1.3000: UDP, length 436
16:25:55.653587 IP 192.168.5.1.21111 > 192.168.210.1.3000: UDP, length 436
```

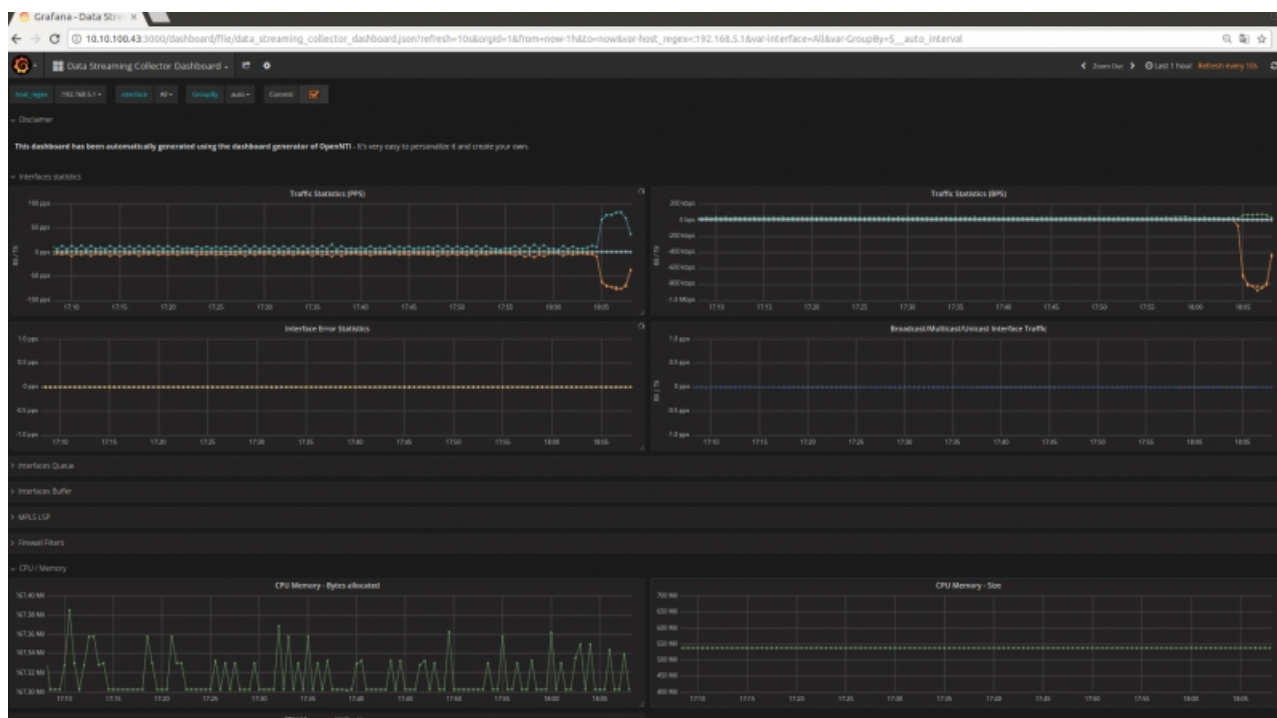
Крайние три блока — настраиваемые сенсоры, в конфигурации которых указываются серверы для отправки данных, export-profile и какие ресурсы подлежат мониторингу. При необходимости, с помощью resource-filter и regexr можно фильтровать необходимые к отправке данные. Перечень ресурсов отражен ниже.

```
root@JTI# set services analytics sensor interface-1 resource ?
```

Possible completions:

```
<resource>    System resource identifier string
/junos/services/cube/usage/
/junos/services/ip-tunnel/usage/ PFE sensor for IP Tunnel statistics
/junos/services/label-switched-path/usage/ PFE sensor for LSP statistics
/junos/services/mgmt-cube/usage/
/junos/system/linecard/cpu/memory/ PFE sensor for CPU (ukernel) memory
/junos/system/linecard/fabric/ PFE sensor for fabric traffic
/junos/system/linecard/firewall/ PFE sensor for firewall filter stats/memory
/junos/system/linecard/interface/ PFE sensor for physical interface traffic
/junos/system/linecard/interface/logical/usage/ PFE sensor for logical interface traffic
/junos/system/linecard/npu/memory/ PFE sensor for NPU memory
/junos/system/linecard/npu/utilization/ PFE sensor for NPU utilization
/junos/system/linecard/optics/ PFE sensor for optics diagnostics
/junos/system/linecard/packet/usage/ PFE sensor for packet statistics
/junos/system/linecard/qmon/ Hardware based queue-monitoring
/junos/system/linecard/services/inline-jflow/ PFE sensor for inline-jflow service stats/memory
```

Наблюдаем данные JTI (дашборд Data Streaming Collector) в Grafana OpenNTI. Всплеск небольшого трафика — запущенный UDP-флуд с помощью iperf.



Резюмируя, можно сказать, что OpenNTI — очень гибкое и масштабируемое решение для сбора и визуализации метрик с устройств сети, способное работать по старой и новой моделям, с возможностью применения в мультивендорных сетях.

