

Consuming XR Model Driven Streaming Telemetry Lab v1

Last Updated: 23-APRIL-2018

IMPORTANT: This content includes pre-release software, and you may experience issues with some features. The included documentation was not created or verified by dCloud. Check Cisco dCloud regularly for new releases!

About This Demonstration

This lab provides a practical approach to consume XR Model Driven Streaming telemetry, using open-source consumers in an automated environment. After exploring how to configure XR routers for Model Driven Streaming telemetry, the lab introduces a consumption pipeline based on InfluxDB, Grafana and Kapacitor – to store, render and alarm on the data being streamed. The last section of the lab introduces Apache Kafka Pub/Sub bus, to distribute the telemetry data to multiple subscribers and explains how to code a basic Python subscriber.

Finally, if you like this lab but you want a personal environment to customize, or create a POC environment in your lab, check the [What Next](#) section.

This guide for the preconfigured Model Streaming Telemetry environment, includes:

- [Requirements](#)
- [About This Solution](#)
- [Topology](#)
- [Users Session](#)
- [Customization Options](#)
- [Get Started](#)
- [Section 1: Understand Model Driven Telemetry](#)
- [Section 2: Configure gRPC Dialout](#)
- [Section 3: Configure gRPC Dialin](#)
- [Section 4: Staging a Telemetry Stack](#)
- [Section 5: Understanding InfluxDB](#)
- [Section 6: Exploring InfluxDB APIs](#)
- [Section 7: Exploring Grafana - My first dashboard](#)
- [Section 8: Exploring the environment reference dashboard](#)
- [Section 9: Exploring near real time measurements using Ostinato](#)
- [Section 10: Streaming Routing Metrics](#)
- [Section 11: Exploring Kapacitor](#)
- [Section 12: Understanding metrics.json](#)

- [Section 13: Troubleshooting YANG model data](#)
- [Section 14: Apache Kafka Pub/Sub bus](#)
- [Section 15: Cleaning up](#)
- [Section 16: What Next](#)

This lab suits beginners, starting with the XR streaming telemetry and progressing step by step, or somebody that has already explored some of basic concepts and would like to focus on a specific concept.

You may decide to follow the sections sequentially (strongly suggested for a beginner), or after completing the initial four sections (that explain and stage the environment), you may freely jump to any of the remaining sections.

Requirements

The table below outlines the requirements for this preconfigured demonstration.

Table 1. Requirements

Required	Optional
Laptop with Cisco® AnyConnect®	Microsoft Remote Desktop (strongly suggested)

About This Solution

As networks grow in size and complexity, the demand for monitoring performance increases. Whether the goal is better traffic engineering, predictive troubleshooting, proactive remediation, or performance auditing, network operators require increasingly real-time visibility of their network infrastructure and services. Decades-old technologies like SNMP simply cannot meet the demand for the volume and granularity of data required. To get as much data about the network as fast as possible, a new technology is required: **model-driven streaming telemetry**.

Model-driven telemetry (MDT) eliminates the inefficiencies of infrastructure polling by streaming the data directly from a device's internal data structures at periodic intervals. Using this technique, the router can export statistics at many times the rate of SNMP. **Cisco® IOS XR** has supported telemetry since XR 6.0.0. Key innovations in XR 6.1.1 include the ability to stream data using YANG models and use gRPC as a transport mechanism (in addition to raw TCP).

This lab aims to equip Systems Engineers with the skills and knowledge required to create, deploy, and run, a self-contained demonstration of IOS XR Streaming Telemetry, infrastructure Big Data processing, basic analytics, visualization and alerting KPI, by using freely available open source products.

The environment proposed in this lab is a self-contained virtual environment running on Virtualbox and it includes an IOS XRv router and an Ubuntu server. It can be easily replicated in a laptop or lab server for customization.

This lab builds on the concepts introduced in the dCloud “Model-Driven Telemetry on Cisco IOS XR Lab v1”, providing hand-on experience using model-driven-telemetry with different kind of applications to collect, display data and alert on data.

The solution proposed aims to prepare the readers to prototype the usage of model-driven telemetry in their labs, by flattening the initial learning phase with projects that can be customized for specific needs.

Topology

This content includes preconfigured users and components to illustrate the scripted scenarios and features of the solution. Most components are fully configurable with predefined administrative user accounts. You can see the IP address and user account credentials to use to access a component by clicking the component icon in the **Topology** menu of your active session and in the scenario steps that require their use.

Figure 1. dCloud Topology

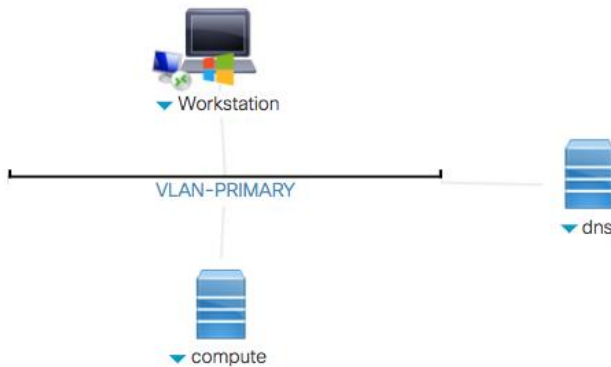
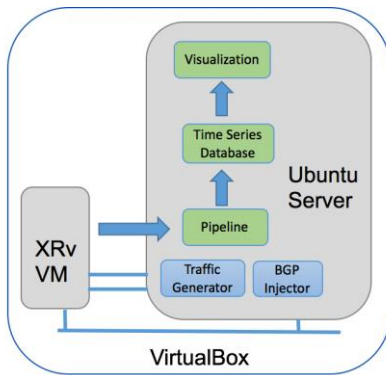


Table 2. Equipment Details

Workstation	Windows 7	198.18.133.252	administrator	C1sco12345
Compute	Ubuntu Server	198.18.128.101	labuser1	C1sco12345

Figure 2. Virtual environment topology



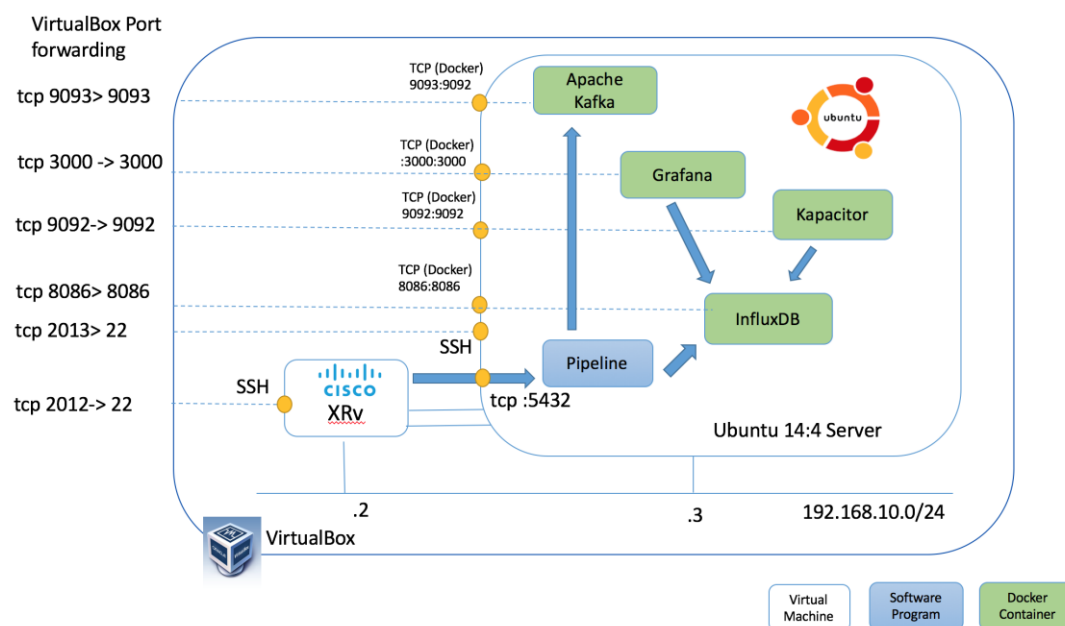
NOTE: The Compute node is configured for nested virtualization using VirtualBox to run an instance of XRv and a Linux device that hosts the telemetry stack. If you decide to deploy this lab on your own infrastructure, these two devices will be deployed in your local VirtualBox environment.

The table below contains details on the environment connectivity (IP addresses and TCP ports).

Table 3. Software connectivity details

Function	Component	Native TCP	Exposed TCP port
Router	SSH to IOS XR router	22	2012
Server	SSH to Ubuntu server	22	2013
Collector	Pipeline	5432	5432
Visualization tool	Grafana	3000	3000
Time Based Database	InfluxDB API	8086	8086
Threshold analysis	Kapacitor	8082	8082
Pub/Sub bus	Apache Kafka	8082	8083
Traffic Generator	Ostinato (Drone)	7878	7878

Figure 3. Environment Detailed Topology and Exposed TCP Ports



Session Users

The table below contains details on preconfigured users available for your session.

Table 4. User Details

User Description	User Name	User Password	User Description
Administrator	administrator	C1sco12345	Windows VM
Lab User	labuser1	C1sco12345	Host Ubuntu
Vagrant User	vagrant	vagrant	XRv router and Programmability Ubuntu Vm (VirtualBox VMs running in the Host Ubuntu)

Customization Options

This lab can be heavily customized by changing the proposed configuration templates (as explained in the guide), by creating new dashboard or different KPIs. This guide has been prepared to provide enough information to decompose the components in the solutions and automation tasks to easy personalize.

If customization of specific use cases is your final goal, you should consider [section 16 – what next](#) for a personal environment.

IMPORTANT NOTE – This lab has been configured to store all received measurements unprocessed in a text file, that you can leverage for troubleshooting as you progress through the document. This feature has the side effect of filling fast the available disk space and if you are planning to run this demo for more than 24 yours or create your own POC, you should consider one of the options suggested in this [workaround](#).

Get Started

BEFORE PRESENTING

Cisco dCloud strongly recommends that you perform the tasks in this document with an active session before presenting in front of a live audience. This will allow you to become familiar with the structure of the document and content.

It may be necessary to schedule a new session after following this guide in order to reset the environment to its original configuration.

PREPARATION IS KEY TO A SUCCESSFUL PRESENTATION.

Follow the steps to schedule a session of the content and configure your presentation environment.

1. Initiate your dCloud session. [\[Show Me How\]](#)

NOTE: It may take up to 15 minutes for your session to become active.

2. For best performance, connect to the workstation with **Cisco AnyConnect VPN** [\[Show Me How\]](#) and the **local RDP client on your laptop** [\[Show Me How\]](#)
 - Workstation 1: **198.18.133.252**, Username: **Administrator**, Password: **C1sco12345**

Section 1: Understand Model Driven Telemetry

This scenario demonstrates the XR model-driven network configuration and validation. More specific, you will:

- Understand the environment proposed in this lab and the provided utilities.
 - Understand the underlying YANG model that determines the data to be streamed.
 - Configure a router for telemetry using Self-describing Google Protocol Buffers (GPB) encoding over TCP.
 - Use a Cisco collector application (Pipeline) to collect and view received telemetry data.
1. Double click on “Telemetry VM” icon to log on the Ubuntu server that will collect the telemetry data.



2. Type `./telemetry_utility.sh` to manage the lifecycle of the open source components that terminate (Pipeline) and consume (Influx, Grafana, Kapacitor and Kafka) the streamed telemetry data.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ./telemetry_utility.sh

1) Verify Environment
2) Destroy & Clean Environment
3) Start/Stop Pipeline
4) Re-start Pipeline
5) Start/Stop Influx, Grafana & Kapacitor
6) Start/Stop Kafka
7) Toggle Pipeline dump logging
8) Truncate Pipeline dump logging
9) Exit
Please enter your choice:
```

NOTE: this document describes the telemetry utility implementation later in the section but for a preview, just type `cat telemetry_utility.sh`. As you can see, most actions call ansible-playbooks to instantiate a configuration state.

3. Type 1 (Verify Environment) at the proposed menu to verify the no telemetry service is currently running on the system.

```
Please enter your choice: 1

!!! Currently checking for Pipeline, Grafana and Influxdb !!!

Checking Pipeline ... down

    Input Stage: TCP Dial-Out
    Input Stage: gRPC Dial-Out
    Output Stage: TAP dump.txt

Checking Grafana ... down

Checking Influxdb ... down
```

```

Checking Kapacitor ... down
Checking Kafka ... down
Checking Zookeeper ... down
Please enter your choice:

```

4. Type 3 (Start/Stop Pipeline) at the proposed menu, confirm your action by pressing any key and enter `vagrant` as vault's password, to start the Pipeline service.

NOTE: user and sudo passwords have been saved in the Ansible vault during the installation and `vagrant` was selected as password to retrieve these information from the vault. Ansible's vault is an effective way to store your secrets locally.

```

Please enter your choice: 3

!!! Ready to START Pipeline !!!

Press any key to continue or Ctrl+C to exit...

Vault password:

PLAY [server]
*****

TASK [Retrieving secrets]
*****
ok: [server]

TASK [Pre-staging phase - including configuration variables]
*****
ok: [server]

TASK [Create data directory /home/vagrant/data]
*****
changed: [server]

TASK [Create log directory /home/vagrant/log]
*****
changed: [server]

TASK [Create dump.txt log file (required if you enable dump in pipeline.conf)]
*****
changed: [server]

TASK [Create pipeline init service (/etc/init/pipeline.conf)]
*****
changed: [server]

TASK [Start Pipeline as init service]
*****
changed: [server]

PLAY RECAP
*****

```

```
server : ok=7 changed=5 unreachable=0 failed=0
```

NOTE: if you are not familiar with Ansible-playbooks, it is a recipe to declare the state for the system and in this case for having Pipeline configured and running correctly. A playbook consists of a long list of tasks and parameters, that calls specialized Ansible module to achieve an outcome (http://docs.ansible.com/ansible/latest/modules_by_category.html).

Using Ansible modules you don't need to care about the requested action implementation. Also, by using detailed task name, a playbook provides a good level of documentation.

A playbook will report a task as "changed" when an action was performed on the system or "ok" when the requested status is already fulfilled. For example, if we would run this playbook a second time, the creation task for the log and data directories would be marked "ok" instead of changed. This introduces a key feature of Ansible which is idempotency: the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening act.

To explore this playbook or any of the other playbooks in the same directory that the telemetry utility invokes, issue the command `cat ~/environment/ansible/start_pipeline.yml` after exiting from the utility.

5. Type 1 (Verify Environment) at the proposed menu, confirm that Pipeline is now running.

```
Please enter your choice: 1

!!! Currently checking for Pipeline, Grafana and Influxdb !!!

Checking Pipeline ... running

Input Stage: TCP Dial-Out
Input Stage: gRPC Dial-Out
Output Stage: TAP dump.txt
```

NOTE: the telemetry utility reports the current status for Pipeline and also the Input, Output and Inspect stages configured for the process (more details later in the section).

6. Type 9 (Exit) at the proposed menu to exit from the telemetry utility.

```
Please enter your choice: 9
BYE
```

7. Type `service pipeline status` to manually confirm that the Pipeline process is actually running.

```
vagrant@vagrant-ubuntu-trusty-64:~$ service pipeline status
pipeline start/running, process 13664
```

Note: if using `ps -ef | grep pipeline` to check the actual running process, you are going to find a first pipeline process owned by root (started by init, process id 1) and a second pipeline process, owned by our user vagrant and started by the initial pipeline process. The vagrant's pipeline process is the actual worker and it was created because (as described in the next step) the init configuration file specifies to run this application as vagrant user (`env APPUSER="vagrant"`).

8. Type `cat /etc/init/pipeline.conf` to visualize Init's configuration to start Pipeline service at boot time and to identify the location of Pipeline's configuration file and logs.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat /etc/init/pipeline.conf
author "marumer@cisco.com"
description "start and stop Pipeline ( using upstart )"
version "1.0"
```



```

start on started networking
stop on runlevel [!2345]

env APPUSER="vagrant"
env APPDIR="/pipeline/bin"
env APPBIN="pipeline"
env APPARGS="-config ~/environment/pipeline.conf -log ~/log/pipeline.log -pem ~/environment/pig"

respawn

script
  exec su - $APPUSER -c "$APPDIR/$APPBIN $APPARGS"
end script

```

Pipeline is an open-source project created by Cisco to consume telemetry streams directly from the routers (or indirectly from a pub/sub bus). Once collected, *pipeline* can perform some limited transformations of the data and forwards the resulting content on to a downstream, typically off-the-shelf, consumers (such as to a text file, Kafka bus, Prometheus, InfluxDB). Pipeline is written in Golang and it can be launched calling the bin/pipeline executable after cloning the GIT repository (<https://github.com/cisco/bigmuddy-network-telemetry-pipeline>). Pipeline expects the location for its configuration file pipeline.conf (passed with -config option) and log file (passed with the -log option).

9. Type `cat ~/environment/pipeline.conf` to inspect the content of the current Pipeline configuration file. This initial configuration starts the TCP input stage (answering on TCP port 5432), the gRPC (Dialout) input stage (answering on TCP port 57500) and the local output tap stage (saving the received metric on the local file `~/log/dump.tx` as clear text).

```

vagrant@vagrant-ubuntu-trusty-64:~$ cat ~/environment/pipeline.conf
[TCPDialout]                                # Local name for the stage
stage = xport input                          # Input stage type
type = tcp                                  # TCP transport
encap = st                                  # streaming telemetry
listen = :5432                               # TCP port Pipeline is listening for this input stage

[gRPCDialout]                               # Local name for the stage
stage = xport input                          # Input stage type
type = grpc                                 # grpc transport
encap = gpb                                 # gpb encapsulation (works for key/value and proto compressed)
listen = :57500                             # TCP port Pipeline is listening for this input stage
tls = false                                 # Configured to work without TLS

#<!-- BEGIN Ansible Managed - Dump Config -->
[inspector]                                 # Local name for the stage
stage = xport output                         # Output stage
type = tap                                  # type TAP that save received streams on local file
file = log/dump.txt                          # location of local file - must be created manually
datachanneldepth = 1000                     # default value
#<!-- END Ansible Managed - Dump Config -->

```

NOTE: The inspector stage is wrapped around `#<!-- ... -->` tags that are used by the Ansible module `blockinfile` (in the playbook activated by the telemetry utility) as anchors to add and remove templates from a file.

10. Double click on "Telemetry VM" icon to create a new SSH session that we are going to use to tail the Pipeline log file.



11. Type `tail -f ~/log/pipeline.log` in the newly created session, to visualize interactively the log generated by Pipeline.

```
vagrant@vagrant-ubuntu-trusty-64:~$ tail -f ~/log/pipeline.log
INFO[2017-12-07 00:13:06.075744] Conductor says hello, loading
config          config="/home/vagrant/environment/pipeline.conf" debug=false fluentd=
logfile="/home/vagrant/log/pipeline.log" maxthreads=1 tag=pipeline.vagrant-ubuntu-trusty-64
version="v1.0.0 (bigmuddy)"
INFO[2017-12-07 00:13:06.076644] Conductor starting up section                name=conductor
section=inspector stage="xport_output" tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:13:06.076702] Conductor starting up section                name=conductor
section=tcpdialout stage="xport input" tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:13:06.076901] Conductor starting up section                name=conductor
section=grpcdialout stage="xport input" tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:13:06.077174] Metamonitoring: not enabled                tag=pipeline.vagrant-
ubuntu-trusty-64
INFO[2017-12-07 00:13:06.077254] Starting up tap                            countonly=false
filename="log/dump.txt" name=inspector streamSpec=&{2 <nil>} tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:32:29.202506] gRPC starting block                        encap=gpb
name=grpcdialout server=:57500 tag=pipeline.vagrant-ubuntu-trusty-64 type="pipeline is SERVER"
INFO[2017-12-07 00:32:29.202641] gRPC: Start accepting dialout sessions    encap=gpb
name=grpcdialout server=:57500 tag=pipeline.vagrant-ubuntu-trusty-64 type="pipeline is SERVER"
INFO[2017-12-07 00:32:29.202703] TCP server starting                       listen=":5432"
name=tcpdialout tag=pipeline.vagrant-ubuntu-trusty-64
```

NOTE: As specified in `pipeline.conf`, the log output confirms that the TCP and gRPC Dial-out services are ready to terminate session and that the metric will be saved in the `log/dump.txt` by the inspector stage.

12. Move the SSH session that is tailing the Pipeline log, on the side and double click on "Telemetry VM" icon again to create a third SSH session.



13. Type `tail -f ~/log/dump.txt` in the newly created session, to visualize interactively the content of the `dump.txt` file and mode this session on the side.

```
vagrant@vagrant-ubuntu-trusty-64:~$ tail -f ~/log/dump.txt
```

NOTE: Don't worry if you don't see any data because the router hasn't been configured yet to stream telemetry. We are going to see the metrics start flowing, as we finish configuring the router (in the next few steps).

NOTE: XR MDT is configured to stream metrics for a specific YANG model path (called Sensor-Path in XR). You can download the supported YANG models from public repositories (i.e. YangModel) and use the `pyang` utility to explore the models and specific paths of interest.

14. Back to our initial SSH session to Telemetry VM (or open a new one, if you close it), change directory to

`~/YangModels/vendor/cisco/xr/613` and issue the command `pyang -f tree Cisco-IOS-XR-infra-statsd-oper.yang --tree-depth 4`, to explore the interface statistic YANG model.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cd ~/YangModels/vendor/cisco/xr/613
vagrant@vagrant-ubuntu-trusty-64:~/YangModels/vendor/cisco/xr/613$ pyang -f tree Cisco-IOS-XR-infra-statsd-oper.yang --tree-depth 4
module: Cisco-IOS-XR-infra-statsd-oper
  +--ro infra-statistics
    +--ro interfaces
      +--ro interface* [interface-name]
        +--ro cache
          | ...
        +--ro latest
          | ...
        +--ro total
          | ...
        +--ro interface-name          xr:Interface-name
        +--ro protocols
          | ...
        +--ro interfaces-mib-counters
```

Note: pyang support the option `--tree-depth` to trim the tree at the specified level and `--tree-path` to filter the path. For example, I am interested in the latest/generic-counters and I want to see more about this specific container.

15. Issue the command `pyang -f tree Cisco-IOS-XR-infra-statsd-oper.yang --tree-depth 6 --tree-path infra-statistics/interfaces/interface/latest/generic-counters` to explore a specific branch of the YANG model, at a lower depth.

NOTE: be careful to use `--tree-depth` and `--tree-path`, with a double dash "--" instead of a single dash "-" because Word often reformats them.

```
vagrant@vagrant-ubuntu-trusty-64:~/YangModels/vendor/cisco/xr/613$ pyang -f tree Cisco-IOS-XR-infra-statsd-oper.yang --tree-depth 6 --tree-path infra-statistics/interfaces/interface/latest/generic-counters
module: Cisco-IOS-XR-infra-statsd-oper
  +--ro infra-statistics
    +--ro interfaces
      +--ro interface* [interface-name]
        +--ro latest
          +--ro generic-counters
            +--ro packets-received?          Uint64
            +--ro bytes-received?            Uint64
            +--ro packets-sent?              Uint64
            +--ro bytes-sent?                Uint64
            +--ro multicast-packets-received? Uint64
            +--ro broadcast-packets-received? Uint64
            +--ro multicast-packets-sent?    Uint64
            +--ro broadcast-packets-sent?    Uint64
            +--ro output-drops?              Uint32
            +--ro output-queue-drops?        Uint32
```

NOTE: Using pyang, I am able to explore the structure of a YANG model and create the MDT Sensor-Path by combining the YANG model name with the tree path that I want to export. In our example: **Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters**, that we will use to configure the XRv router in the next steps.

Using pyang, we have explored the YANG model structure, in [Section 13: Troubleshooting YANG model data](#), we are going to validate the actual data contained in a YANG path.

16. Change directory to ~ (user's "home" folder), issue the command `./XR_demo.sh` and choose option 1 (from the proposed menu) to confirm that there is no active model driven telemetry configuration on the test router.

```
$ vagrant@vagrant-ubuntu-trusty-64:~/YangModels/vendor/cisco/xr/613$ cd ~
$ vagrant@vagrant-ubuntu-trusty-64:~$ ./XR_demo.sh

1) Retrieve current XR telemetry configuration
2) Remove XR telemetry configuration
3) Configure example Sensor-Group
4) Configure example Destination-Group
5) Configure example Subscription
6) Apply MDT template for Dial-out, TCP and GPBkv
7) Apply MDT template for Dial-out, gRPC without TLS and GPBkv
8) Apply MDT template for Dial-in, gRPC without TLS and GPBkv
9) Add and remove Dial-In from Pipeline
10) Exit
Please enter your choice: 1

PLAY [Routers]
*****

TASK [Executing: show running-config telemetry model-driven]
*****
ok: [XR_test_192.168.10.2]

TASK [Output for: show running-config telemetry model-driven]
*****
ok: [XR_test_192.168.10.2] => {
  "config.stdout lines": [
    [
      "% No such configuration item(s)"
    ]
  ]
}

PLAY RECAP
*****
XR test 192.168.10.2      : ok=2    changed=0    unreachable=0    failed=0

Please enter your choice:
```

NOTE: The ansible playbook `router_command.yml` used in this step, displays the output of `show running-config telemetry model-driven`. If you prefer to run commands in CLI, just double click on the `XR VM` icon and type them by hand.

17. Now choose option 3, to configure an example Sensor Group. After reviewing the proposed template, press return to configure the router with the example sensor-group Sgroup101 and the associated sensor-paths.

NOTE: Remember that a sensor-groups describes **WHAT** should be streamed in a policy.

Model Driven Telemetry leverages YANG models for this task and a sensor-path represents a container of operational data contained in the specified model's branch. Each sensor-group has one or more sensor-paths.

This example will export common metrics like CPU utilization, memory utilization, interface statistics, processes statistics and an example of routing information (BGP) as a sensor-group named Sgroup101.

Please enter your choice: 3

!!! Configuring example of Sensor-Group !!!

PLAY [Routers]

TASK [Ready to apply XR conf/sensor-group.cfg]

```
ok: [XR test 192.168.10.2] => {
  "msg": [
    "telemetry model-driven",
    " sensor-group Sgroup101",
    "  sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
    "  sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
    "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
    /latest/data-rate",
    "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
    /latest/generic-counters",
    "  sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
    /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information"
  ]
}
```

TASK [pause]

[pause]

Press Return key to confirm - To abort a playbook press ctrl+c and then a:

ok: [XR test 192.168.10.2]

TASK [Applying template XR conf/sensor-group.cfg]

changed: [XR test 192.168.10.2]

TASK [Retrieve current MDT configuration]

ok: [XR_test_192.168.10.2]

TASK [Print current MDT Configuration]

```
ok: [XR test 192.168.10.2] => {
  "config.stdout_lines": [
```

```

[
  "telemetry model-driven",
  " sensor-group Sgroup101",
  "  sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
  "  sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
  "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
    /latest/data-rate",
  "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
    /latest/generic-counters",
  "  sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
    /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
  " !",
  "!"
]
]
}

```

PLAY RECAP

XR_test_192.168.10.2 : ok=5 changed=1 unreachable=0 failed=0

Please enter your choice:

NOTE: How does it work? The `XR_demo.sh` shell utility, calls the Ansible playbook

`~/environment/ansible/router_template.yml` passing the `XR_conf/sensor-group.cfg` option that is a CLI template stored in `~/environment/ansible/XR conf`.

The Ansible playbook configured the XRv router with the proposed template, using the standard Ansible module `iosxr_config` and retrieves the new configuration from the router using the Ansible module `iosxr_command`.

You may consider these modules when planning some basic template driven automation for your XR routers.

18. Double click on the XR VM icon to open an SSH session to XRv router under test.



19. Type `show telemetry model-driven sensor-group` in the router CLI, to confirm that the configured sensor-paths are validated against the supported YANG's models.

```
RP/0/RP0/CPU0:test_XR#show telemetry model-driven sensor-group
```

```
Mon Nov 27 05:22:40.891 UTC
```

```
Sensor Group Id:Sgroup101
```

```
Sensor Path: Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
```

```
Sensor Path State: Resolved
```

```
Sensor Path: Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary
```

```
Sensor Path State: Resolved
```

```
Sensor Path: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
/latest/data-rate
```

```
Sensor Path State: Resolved
```

```
Sensor Path: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
/latest/generic-counters
```

```

Sensor Path State: Resolved
Sensor Path:      Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names
                  /ip-rib-route-table-name/protocol/bgp/as/information
Sensor Path State: Resolved
Deleting pre-configured XR Model Driven Telemetry section.

```

NOTE: When reporting a `Resolved` state, this troubleshooting command confirms the consistency for a configured sensor path against a YANG model tree. [Later](#) in the document, we will validate the data stored in the model.

20. Back to the Telemetry VM session, choose now options 4 (from the `XR_demo.sh` utility that should be still running) to configure an example of Destination Group. After reviewing the proposed template, press return to configure the router.

NOTE: Remember that a destination-group that describes **WHERE** to stream and **HOW** to encode the telemetry streams. The destination group specifies the destination address, port, encoding and transport that the router uses to send out telemetry data. In this case, we are configuring the router to stream telemetry as TCP Dialout (encoding as self-describing GPB) toward the Telemetry VM we have configured in the initial steps of this section.

```

Please enter your choice: 4

!!! Configuring example of destination-group !!!

PLAY [Routers]
*****

TASK [Ready to apply XR_conf/destination-group.cfg]
*****
ok: [XR_test_192.168.10.2] => {
  "msg": [
    "telemetry model-driven",
    " destination-group DGroup1",
    " address-family ipv4 192.168.10.3 port 5432",
    " encoding self-describing-gpb",
    " protocol tcp",
    " !",
    " !"
  ]
}

TASK [pause]
*****
[pause]
Press Return key to confirm - To abort a playbook press ctrl+c and then a:

ok: [XR_test_192.168.10.2]

TASK [Applying template XR_conf/destination-group.cfg]
*****
changed: [XR test 192.168.10.2]

TASK [Retrieve current MDT configuration]
*****
ok: [XR test 192.168.10.2]

TASK [Print current MDT Configuration]
*****

```

```
ok: [XR test 192.168.10.2] => {
  "config.stdout lines": [
    [
      "telemetry model-driven",
      " destination-group DGroup1",
      "  address-family ipv4 192.168.10.3 port 5432",
      "    encoding self-describing-gpb",
      "    protocol tcp",
      "  !",
      "  !",
      " sensor-group Sgroup101",
      "  sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
      "  sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
      "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
        /latest/data-rate",
      "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
        /latest/generic-counters",
      "  sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
        /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
      "  !",
      "  !"
    ]
  ]
}
```

PLAY RECAP

```
*****
XR test 192.168.10.2      : ok=5    changed=1    unreachable=0    failed=0
```

21. Finally choose options 5 from the proposed menu, to configure an example of Subscription. After reviewing the proposed template, press return to configure the router. This will configure a subscription with our sensor-group Sgroup101 data, which exports to our destination-group Dgroup1.

NOTE: Remember that a telemetry subscription links together sensors (WHAT), destinations (WHERE), encoding (HOW) with cadence (WHEN), by binding together one or more sensor-groups with one of more destination-groups.

Please enter your choice: 5

!!! Configuring example of subscription !!!

PLAY [Routers]

```
*****
```

TASK [Ready to apply XR conf/subscription.cfg]

```
*****
```

```
ok: [XR_test_192.168.10.2] => {
  "msg": [
    "telemetry model-driven",
    " subscription 1",
    "  sensor-group-id Sgroup101 sample-interval 1000",
    "  destination-id DGroup1"
  ]
}
```



```

TASK [pause]
*****
[pause]
Press Return key to confirm - To abort a playbook press ctrl+c and then a:

ok: [XR test 192.168.10.2]

TASK [Applying template XR conf/subscription.cfg]
*****
changed: [XR_test_192.168.10.2]

TASK [Retrieve current MDT configuration]
*****
ok: [XR test 192.168.10.2]

TASK [Print current MDT Configuration]
*****
ok: [XR test 192.168.10.2] => {
  "config.stdout_lines": [
    [
      "telemetry model-driven",
      " destination-group DGroup1",
      "  address-family ipv4 192.168.10.3 port 5432",
      "    encoding self-describing-gpb",
      "    protocol tcp",
      "  !",
      " !",
      " sensor-group Sgroup101",
      "   sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
      "   sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
      "   sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
        /latest/data-rate",
      "   sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
        /latest/generic-counters",
      "   sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
        /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
      "  !",
      " subscription 1",
      "   sensor-group-id Sgroup101 sample-interval 1000",
      "   destination-id DGroup1",
      "  !",
      " !"
    ]
  ]
}

PLAY RECAP
*****
XR_test_192.168.10.2      : ok=5    changed=1    unreachable=0    failed=0

```

IMPORTANT NOTE: this configuration uses a 1 second sample-interval (1000 msec) to demonstrate near real-time support. At the time of writing, XR 6.2.x release notes, confirm support up to 10 seconds and you should always consider the suitability for low sample values, based on the size of the data bag to stream. Check with the Cisco Engineering or your account team, before considering lower values.

22. Switch back to the SSH session that is tailing the pipeline.log file, to confirm a new incoming session for the configured subscription.

```
vagrant@vagrant-ubuntu-trusty-64:~$ tail -f log/pipeline.log
INFO[2017-12-07 00:32:29.200337] Conductor says hello, loading
config      config="/home/vagrant/environment/pipeline.conf" debug=false fluentd=
logfile="/home/vagrant/log/pipeline.log" maxthreads=1 tag=pipeline.vagrant-ubuntu-trusty-64
version="v1.0.0 (bigmuddy)"
INFO[2017-12-07 00:32:29.201098] Conductor starting up section          name=conductor
section=inspector stage="xport output" tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:32:29.201156] Conductor starting up section          name=conductor
section=grpcdialout stage="xport_input" tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:32:29.201273] Conductor starting up section          name=conductor
section=tcpdialout stage="xport_input" tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:32:29.202332] Metamonitoring: not enabled          tag=pipeline.vagrant-
ubuntu-trusty-64
INFO[2017-12-07 00:32:29.202410] Starting up tap                  countonly=false
filename="log/dump.txt" name=inspector streamSpec=&{2 <nil>} tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 00:32:29.202506] gRPC starting block              encap=gpbb
name=grpcdialout server=:57500 tag=pipeline.vagrant-ubuntu-trusty-64 type="pipeline is SERVER"
INFO[2017-12-07 00:32:29.202641] gRPC: Start accepting dialout sessions          encap=gpbb
name=grpcdialout server=:57500 tag=pipeline.vagrant-ubuntu-trusty-64 type="pipeline is SERVER"
INFO[2017-12-07 00:32:29.202703] TCP server starting              listen=":5432"
name=tcpdialout tag=pipeline.vagrant-ubuntu-trusty-64

INFO[2017-12-07 02:19:02.446608] TCP server accepted connection          encap=st keepalive=0s
local="192.168.10.3:5432" name=tcpdialout remote="192.168.10.2:25699" tag=pipeline.vagrant-ubuntu-
trusty-64
```

23. Switch to the SSH session that is tailing the dump.txt, to confirm a continuous flow of metrics being stored locally. You may need to stop to continuous flow of metrics using CTRL-C and issue the `tail -f log/dump.txt` again, when you have finished checking the file output.

IMPORTANT NOTE – As we have already discussed that Pipeline has been configured in this lab, with the TAP output function and it will store all received measurements unprocessed in the `/home/vagrant/log/dump.txt` file.

`dump.txt` a text file and the unprocessed measurements will consume a lot of disk, filling the available space when this demo is left storing data for more a day.

If you are planning to collect data for more than 24 hours demonstration or test, you should consider disabling the local dumping of unprocessed measurements using the following procedure:

- Double click on **Telemetry VM** icon to log on the Ubuntu server that will collect the telemetry data.
- Type `./telemetry utility.sh` to manage the lifecycle of the open source components that terminates (Pipeline) and consumes (Influx, Grafana, Kapacitor and Kafka) the streamed telemetry data.
- Type 7 (**Toggle Pipeline dump logging**) at the proposed menu to verify and enable/disable the current dumping of unprocessed measurements.
- Finally, type 9 (**Exit**) to exit the utility and exit at the SSH prompt to close the session

Alternatively if you don't want to disable this functionality, you may use option 8 (**Truncate Pipeline dump logging**) in the same `./telemetry utility.sh` every 24 hours to trim the length of `/home/vagrant/log/dump.txt` to null, without impacting the logging activity.

Finally, if you are planning to deploy Pipeline in a POC and you want to keep active the dumping functionality, you can also consider managing `/home/vagrant/log/dump.txt` with `logrotate` (outside the scope of this demo but you can check the examples in the `/etc/logrotate.d/` directory).

```
vagrant@vagrant-ubuntu-trusty-64:~$ tail -f log/dump.txt
```

```
----- 2017-12-07 03:00:09.075700631 +0000 UTC -----
```

```
Summary: GPB(common) Message [192.168.10.2:25699(test_XR)/Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-name/protocol/bgp/as/information msg len: 562]
```

```
{
  "Source": "192.168.10.2:25699",
  "Telemetry": {
    "node_id_str": "test_XR",
    "subscription_id_str": "1",
    "encoding_path": "Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
    "collection_id": 3354519,
    "collection start time": 1512615609007,
    "msg timestamp": 1512615609008,
    "collection end time": 1512615609073
  },
  "Rows": [
    {
      "Timestamp": 1512615609053,
      "Keys": {
        "af-name": "IPv4",
        "as": "65000",
        "route-table-name": "default",
        "saf-name": "Unicast",
        "vrf-name": "default"
      },
      "Content": {
        "active-routes-count": 0,
        "backup-routes-count": 0,
        "deleted-routes-count": 0,
        "instance": "65000",
        "paths-count": 0,
        "protocol-clients-count": 1,
        "protocol-names": "bgp",
        "protocol-route-memory": 0,
        "redistribution-client-count": 0,
        "routes-counts": 0,
        "version": 0
      }
    }
  ]
}
```

```
----- 2017-12-07 03:00:09.112628097 +0000 UTC -----
```

```
Summary: GPB(common) Message [192.168.10.2:25699(test_XR)/Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization msg len: 40125]
```

```
{
  "Source": "192.168.10.2:25699",
  "Telemetry": {
    "node_id_str": "test_XR",
```

```

    "subscription id str": "1",
    "encoding path": "Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
    "collection id": 3354520,
    "collection start time": 1512615609074,
    "msg timestamp": 1512615609074,
    "collection end time": 1512615609088
  },
<SNIP>

```

NOTE: storing uncompressed streams in the `dump.txt` files, is an important feature to learn the structure of new measurements or for troubleshooting but remember that the size of the `dump.txt` grows fast. When not needed, you should consider disabling this functionality.

`telemetry_utility.sh` script provides `Toggle Pipeline dump logging`, to enable and disable the dumping function.

24. Switch back to the SSH session to the XR's router and issue the command `show telemetry model-driven subscription`, to confirm that the sensor group is fully resolved and the destination group is active.

```

RP/0/RP0/CPU0:test_XR#show telemetry model-driven subscription
Thu Dec  7 02:29:11.944 UTC
Subscription: 1                               State: ACTIVE
-----
Sensor groups:
Id          Interval(ms)      State
Sgroup101   1000                   Resolved

Destination Groups:
Id          Encoding          Transport  State  Port  IP
DGroup1     self-describing-gpb tcp        Active  5432  192.168.10.3
No TLS :

```

25. Issue `show telemetry model-driven subscription 1 internal` command, if you need to troubleshoot a subscription.

```

RP/0/RP0/CPU0:test_XR#show telemetry model-driven subscription 1 internal
Thu Dec  7 02:30:41.154 UTC
Subscription: 1
-----
State:      ACTIVE
Sensor groups:
Id: Sgroup101
  Sample Interval:      1000 ms
  Sensor Path:          Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
  Sensor Path State:    Resolved
  Sensor Path:          Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary
  Sensor Path State:    Resolved
< SNIP >

Destination Groups:
Group Id: DGroup1
  Destination IP:       192.168.10.3
  Destination Port:     5432
  Encoding:             self-describing-gpb
  Transport:            tcp
  State:                Active
  No TLS
  Total bytes sent:     31720162

```

```

Total packets sent: 3284
Last Sent time: 2017-12-07 02:29:39.3607608605 +0000

Collection Groups:
-----
Id: 26
Sample Interval: 1000 ms
Encoding: self-describing-gpb
Num of collection: 657
Collection time: Min: 1 ms Max: 157 ms
Total time: Min: 12 ms Avg: 26 ms Max: 168 ms
Total Deferred: 0
Total Send Errors: 0
Total Send Drops: 0
Total Other Errors: 0
Last Collection Start: 2017-12-07 02:29:39.3607512605 +0000
Last Collection End: 2017-12-07 02:29:39.3607554605 +0000
Sensor Path: Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization

Sysdb Path: /oper/wdsysmon_fd/gl/*
Count: 657 Method: DATALIST Min: 6 ms Avg: 11 ms Max: 157 ms
Item Count: 657
Missed Collections: 41 send bytes: 26525867 packets: 657 dropped bytes: 0

```

	success	errors	deferred/drops
Gets	0	0	
List	0	0	
Datalist	657	0	
Finddata	0	0	
GetBulk	0	0	
Encode		0	0
Send		0	0

This troubleshooting command provides for each sensor-path, valuable information for each collection cycle, including the time spent to perform the collection, collection errors and the Sysdb Path that internally stores the specified path information (discussed later in [Troubleshooting YANG model data](#)).

Section 2: Configure gRPC Dialout

In this section, we are going to build on the initial setup, by explaining how to change the model driven configuration to use gRPC instead of TCP transmission.

Enabling gRPC may require the configuration of a `Third Party App` (TPA) to provide proper IP connectivity to the gRPC process implemented in the Linux shell of a 64 bit system. Check <https://xrdocs.github.io/telemetry/tutorials/2017-05-05-mdt-with-grpc-transport-tricks> for more information about gRPC connectivity requirements and troubleshooting.

1. Switch back to the initial Telemetry VM session, that should still be running `./XR_demo.sh` (or launch the utility again) and choose options 7, to configure the same model driven telemetry configuration but for gRPC dialout.
After reviewing the proposed template, press return to configure the router.

NOTE: the proposed template for gRPC dialout uses the same sensor-group and subscription, configured in the previous TCP transport example and the only changes to enable gRPC, are in the destination-group.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ./XR_demo.sh

Please enter your choice:
1) Retrieve current XR telemetry configuration
2) Remove XR telemetry configuration
3) Configure example Sensor-Group
4) Configure example Destination-Group
5) Configure example Subscription
6) Apply MDT template for Dial-out, TCP and GPBkv
7) Apply MDT template for Dial-out, gRPC without TLS and GPBkv
8) Apply MDT template for Dial-in, gRPC without TLS and GPBkv
9) Add and remove Dial-In from Pipeline
10) Exit
Please enter your choice: 7

!!! Configuring MDT template for Dial-out, gRPC without TLS and GPBkv !!!
!!! Remove other MDT configurations !!!

PLAY [Routers]
*****

TASK [Ready to apply XR_conf/MDT_GRPc_BPbKv_OUT.cfg]
*****
ok: [XR test 192.168.10.2] => {
  "msg": [
    "tpa",
    " address-family ipv4",
    "  update-source GigabitEthernet0/0/0/0",
    " !",
    "!",
    "",
    "grpc",
    " port 57500",
    "!",
    "",
    "no telemetry model-driven",
    "",
    "telemetry model-driven",
```

```

    " destination-group DGroup1",
    " address-family ipv4 192.168.10.3 port 57500",
    " encoding self-describing-gpb",
    " protocol grpc no-tls",
    " !",
    " !",
    " sensor-group Sgroup101",
    " sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
    " sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
    " sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
      /latest/data-rate",
    " sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
      /latest/generic-counters",
    " sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
      /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
    " !",
    " subscription 1",
    " sensor-group-id Sgroup101 sample-interval 1000",
    " destination-id DGroup1",
    " !",
    " !"
  ]
}

```

TASK [pause]

[pause]

Press Return key to confirm - To abort a playbook press ctrl+c and then a:

ok: [XR test 192.168.10.2]

TASK [Applying template XR_conf/MDT_GRPC_BPBkv_OUT.cfg]

changed: [XR test 192.168.10.2]

TASK [Retrieve current MDT configuration]

ok: [XR_test_192.168.10.2]

TASK [Print current MDT Configuration]

ok: [XR test 192.168.10.2] => {

"config.stdout lines": [

[

```

      "telemetry model-driven",
      " destination-group DGroup1",
      " address-family ipv4 192.168.10.3 port 57500",
      " encoding self-describing-gpb",
      " protocol grpc no-tls",
      " !",
      " !",
      " sensor-group Sgroup101",
      " sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
      " sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
      " sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
        /latest/data-rate",
      " sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface

```

```

        /latest/generic-counters",
        " sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
        /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
        " !",
        " subscription 1",
        " sensor-group-id Sgroup101 sample-interval 1000",
        " destination-id DGroup1",
        " !",
        " !"
    ]
}

PLAY RECAP
*****
XR_test_192.168.10.2      : ok=5    changed=1    unreachable=0    failed=0

```

- Switch to the SSH session that is tailing the pipeline.log file, to confirm the closure of the initial TCP session and the opening of a new incoming session using gRPC. This may take 5-10 seconds to happen.

```

vagrant@vagrant-ubuntu-trusty-64:~$ tail -f log/pipeline.log

<SNIP>
ERRO[2017-12-07 03:45:45.775920] TCP server failed on read full          error=EOF
tag=pipeline.vagrant-ubuntu-trusty-64
INFO[2017-12-07 03:45:45.776028] TCP server closing connection      encap=st keepalive=0s
local="192.168.10.3:5432" name=tcpdialout remote="192.168.10.2:25699" tag=pipeline.vagrant-ubuntu-trusty-64

INFO[2017-12-07 03:45:45.792582] gRPC: Receiving dialout stream      encap=gpb
name=grpcdialout peer="192.168.10.2:61451" server=:57500 tag=pipeline.vagrant-ubuntu-trusty-64
type="pipeline is SERVER"

```

NOTE: the new gRPC session type confirms that this is a dialout session because "Pipeline is SERVER".

- Switch to the SSH session that is tailing the dump.txt, to confirm that the measurements flow has restarted.
- Finally, issue the `show telemetry model-driven subscription` command in the SSH toward the XR router, to confirm Active state of the subscription and validate its details.

```

RP/0/RP0/CPU0:test_XR#show telemetry model-driven subscription
Thu Dec  7 04:02:11.510 UTC
Subscription: 1                               State: ACTIVE
-----
Sensor groups:
Id          Interval(ms)      State
Sgroup101   1000                  Resolved

Destination Groups:
Id          Encoding          Transport  State  Port  IP
DGroup1     self-describing-gpb  grpc       Active 57500 192.168.10.3

```


Section 3: Configure gRPC Dialin

In the previous section, we configured gRPC Dialout by simply changing the configuration for the Destination Group and adjusting the TPA configuration.

Enabling gRPC Dial-in is a bit more involving, we will need to:

- Enable gRPC server side on the router.
 - Configure a `Third Party App` (TPA) that provides IP connectivity to the gRPC server implemented in the Linux shell of a 64 bit system, to reach Pipeline. Check <https://xrdocs.github.io/telemetry/tutorials/2017-05-05-mdt-with-grpc-transport-tricks> for more information about gRPC connectivity requirements and troubleshooting.
 - Create a local router username, with access to model driven telemetry system. You can alternatively use a AAA configuration.
 - Remove the destination-id from the XR router telemetry Subscription configuration.
 - Configure Pipeline to initiate the gRPC session, for a specific Subscription.
1. Switch back to the Telemetry VM session that should still running `./XR_demo.sh` (or launch the utility again) and choose option 8 (Apply MDT template for Dial-in, gRPC without TLS and GPBkv), to configure the same model driven telemetry but for gRPC Dialin.

After reviewing the proposed template, press return to configure the router.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ./XR_demo.sh

1) Retrieve current XR telemetry configuration
2) Remove XR telemetry configuration
3) Configure example Sensor-Group
4) Configure example Destination-Group
5) Configure example Subscription
6) Apply MDT template for Dial-out, TCP and GPBkv
7) Apply MDT template for Dial-out, gRPC without TLS and GPBkv
8) Apply MDT template for Dial-in, gRPC without TLS and GPBkv
9) Add and remove Dial-In from Pipeline
10) Exit
Please enter your choice: 8

!!! Configuring MDT template for Dial-in, gRPC without TLS and GPBkv !!!
!!! Remove other MDT configurations !!!

PLAY [Routers]
*****

TASK [Ready to apply XR conf/MDT GRPC BPBkv IN.cfg]
*****

ok: [XR test 192.168.10.2] => {
  "msg": [
    "tpa",
    " address-family ipv4",
    " update-source GigabitEthernet0/0/0/0",
    " !",
    " !",
  ]
}
```

```

    "",
    "grpc",
    " port 57500",
    "!",
    "",
    "username mdt",
    " group sysadmin ",
    " password telemetry",
    "!",
    "",
    "no telemetry model-driven",
    "",
    "telemetry model-driven",
    " sensor-group Sgroup101",
    "  sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
    "  sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
    "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
    /latest/data-rate",
    "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
    /latest/generic-counters",
    "  sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
    /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
    " ",
    " subscription 1",
    "  sensor-group-id Sgroup101 sample-interval 1000"
  ]
}

TASK [pause]
*****
[pause]
Press Return key to confirm - To abort a playbook press ctrl+c and then a:

ok: [XR test 192.168.10.2]

TASK [Applying template XR conf/MDT GRPC BPBkv IN.cfg]
*****
changed: [XR test 192.168.10.2]

TASK [Retrieve current MDT configuration]
*****
ok: [XR test 192.168.10.2]

TASK [Print current MDT Configuration]
*****
ok: [XR_test_192.168.10.2] => {
  "config.stdout lines": [
    [
      "telemetry model-driven",
      " sensor-group Sgroup101",
      "  sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization",
      "  sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary",
      "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
      /latest/data-rate",
      "  sensor-path Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
      /latest/generic-counters",
      "  sensor-path Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf
    ]
  ]
}

```

```

        /ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
        " !",
        " subscription 1",
        "  sensor-group-id Sgroup101 sample-interval 1000",
        " !",
        "!"
    ]
}
}

PLAY RECAP
*****
XR test 192.168.10.2      : ok=5    changed=1    unreachable=0    failed=0

```

2. Select option 9 (Add and remove Dial-In from Pipeline) to configure Pipeline to open to open a gRPC Dial-in session toward the router. When requested, type `vagrant` as vault password.

```

Please enter your choice: 9

!!! Ready to ADD Dial-out to Pipeline for Dial-in end RE-START the service !!!

Press any key to continue or Ctrl+C to exit...

Vault password:

PLAY [server]
*****

TASK [Retrieving secrets]
*****
ok: [server]

TASK [Pre-staging phase - including configuration variables]
*****
ok: [server]

TASK [Checking Pipeline process status]
*****
ok: [server]

TASK [Update pipeline.conf for GRPC Dial-In]
*****
changed: [server]

TASK [Restart Pipeline (init) - if running]
*****
changed: [server]

PLAY RECAP
*****
server      : ok=5    changed=2    unreachable=0    failed=0

```

3. Select option 10 (Exit) to exit from the XR_demo.sh tool.
4. Issue `cat /home/vagrant/environment/pipeline.conf` to check Pipeline's updated configuration.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat /home/vagrant/environment/pipeline.conf
```

```
[TCPDialout]
stage = xport input
type = tcp
encap = st
listen = :5432

[gRPCDialout]
stage = xport input
type = grpc
encap = gpb
listen = :57500
tls = false

#<!--BEGIN Ansible Managed - Dump Config ->
[inspector]
stage = xport output
type = tap
file = log/dump.txt
datachanneldepth = 1000
#<!--END Ansible Managed - Dump Config ->

#<!--BEGIN Ansible Managed - GRPC Dial-In Config ->
[gRPCDialin]
stage = xport input
type = grpc
encoding = gpbkv
encap = gpb
server = 192.168.10.2:57500
subscriptions = 1
tls = false
# Username mdt and password telemetry - usage of PEM avoid request for username and password
username=mdt
password=DDh68tet1Ilnzpyck5y9Umd26NsoFtt8MWLyUTUSIFmeL7S/XaplOwB/rHPMl6P5WipS9ckruf6zWd5yrRR4jgQc7B9PgFo
V6pJ3DihfpSs2uhlIFBfMcZHKvpGcGoVxP8NmYzeiK4MfQvon4wl+E5bZOYNTToTYYKf3Fn+9IaQigkOWU4ixUcPt6les6mVuNlKq8VOW
J9RjbgEPE5C09vac1BA3MZsJDH7PpbUXuqUBbaPP4Y4ozwyK7jiPhdeeQOiDW/5ba4cK7OJ6A+uMHFFotQJtzhzUuSP2UzEbDyj0AOjJ
tO4XkECov7zcF3pH6v4kD25r1TdwEn0HVCQPr7ObMg0qu77hgQIMEQO8iZ7UXJYuoCX6rDYN4ceB5Xyx+mejLi5Ht7OEVRsUos/YVGx2
rVwTcd6LQbLRmXw1MRQ+G7rReqzcUWGBzClTcQLPkKpizeL7D+RzXT1/uzoKjFyd0al+FPYri22edx1YVPazMIQ3NexB1vGB74EvoV+n
tMb9vFraqnHG5y5cY0i3CBNnnVC7yphIMDzrAaqYlzxmi1KrSNbZ6TxgnneYDojopxQYApYuCOSwjoFOvmg9YYWPF5kOPDPLy9k152c
8NS4nWufDeBZAgrlatDfMOOb3srFopih7bvyIV2OjzRWOHgLyLhUIbTQ50qcElYoq77s=
#<!--END Ansible Managed - GRPC Dial-In Config ->
```

NOTE: when configuring gRPC Dial-in, Pipeline's default configuration is to interactively request the username and password for the remote router. Alternatively, we can launch Pipeline with the `-pem` (local certificate) option, answer the password challenge and store the encrypted password returned in a modified `pipeline.conf` file, for the future dial-in configuration using the same username, password and PEM certificate.

- Switch to the SSH session that is tailing the `pipeline.log` file, to confirm the closure of the initial TCP session and the opening of a new incoming session using gRPC.

```
vagrant@vagrant-ubuntu-trusty-64:~$ tail -f log/pipeline.log
<SNIP>
INFO[2017-12-07 04:55:25.782497] gRPC: Start accepting dialout sessions          encap=gpb
name=grpcdialout server=:57500 tag=pipeline.vagrant-ubuntu-trusty-64 type="pipeline is SERVER"
```

```
INFO[2017-12-07 04:55:25.782796] gRPC starting block encap=gpb
encodingRequest=gpbkv name=grpcdialin server=192.168.10.2:57500 subscriptions=[1] tag=pipeline.vagrant-
ubuntu-trusty-64 type="pipeline is CLIENT" username=mdt
INFO[2017-12-07 04:55:25.783032] gRPC: Connected codec=gpb encap=gpb
encodingRequest=gpbkv name=grpcdialin server=192.168.10.2:57500 subscriptions=[1] tag=pipeline.vagrant-
ubuntu-trusty-64 type="pipeline is CLIENT" username=mdt
INFO[2017-12-07 04:55:25.807967] gRPC: Subscription handler running encap=gpb
encodingRequest=gpbkv name=grpcdialin reqID=2372179681499370836 server=192.168.10.2:57500 subscription=1
subscriptions=[1] tag=pipeline.vagrant-ubuntu-trusty-64 type="pipeline is CLIENT" username=mdt
```

NOTE: the new gRPC session type confirms that this is a Dial-In session because Pipeline is CLIENT.

6. Switch to the SSH session that is tailing the `dump.txt`, to confirm that the measurements flow has restarted.
7. In the SSH session to the XR router, issue the `show telemetry model-driven destination` command, to visualize a dynamic destination-group, created by the Dialin session and its current status.

```
RP/0/RP0/CPU0:test_XR#show telemetry model-driven destination
Thu Dec 7 05:18:27.248 UTC
Group Id      IP          Port      Encoding      Transport      State
-----
DialIn 1008   192.168.10.3  33482        self-describing-gpb dialin  Active
```

NOTE: You can still use any standard troubleshooting command, for example `show telemetry model-driven destination <DialIn_#>`.

8. Issue `show telemetry model-driven subscription` command, to confirm the Active state of the Subscription and the bounding with the dynamic Destination Group created by the Dial-In session.

```
RP/0/RP0/CPU0:test_XR#show telemetry model-driven subscription
Thu Dec 7 05:14:04.868 UTC
Subscription: 1                      State: ACTIVE
-----
Sensor groups:
Id      Interval (ms)      State
Sgroup101  1000              Resolved

Destination Groups:
Id      Encoding      Transport      State      Port      IP
DialIn 1008   self-describing-gpb dialin  Active  33482  192.168.10.3
No TLS :
```

NOTE: for the next sessions, we are going to leave the router configured for gRPC Dial-In but any other configuration (gRPC Dial-Out or TCP transport) would in the same way.

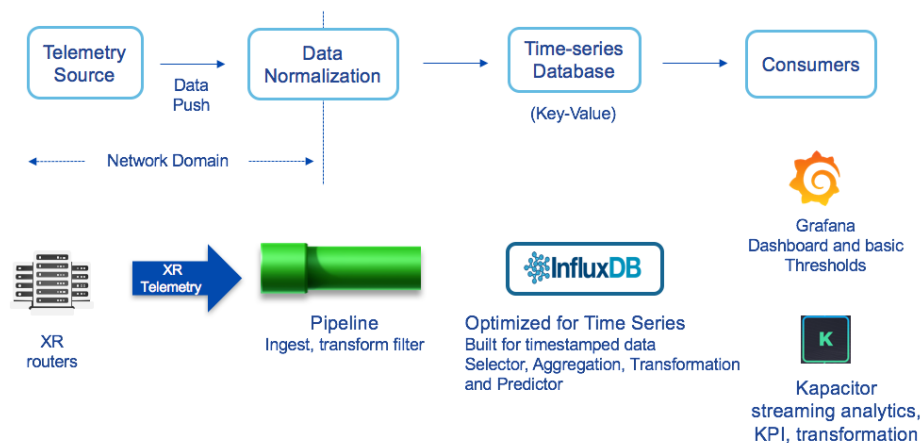
9. CTRL-C in any SSH session any still running the `tail -f` command and close all SSH sessions.

Section 4: Staging a Telemetry Stack

This session introduces the concept of streaming telemetry consumers that process the data generated by Pipeline collector's output stages. In particular, we are going to start and explain the environment automation, leaving to the next few chapters the details on working with InfluxDB, using InfluxDB API, creating an initial Grafana dashboard, using the environment reference dashboard and finally configuring Kapacitor, with an example of CPU utilization KPI alarm.

Based on your level of experience, interest in following this lab and time availability, you can decide to skip some of these sections but you should complete this section (to create the telemetry reference environment) and section 8 "[Exploring a more advanced Grafana dashboard](#)" because this document refers many times to the environment reference dashboard.

Figure 4. Telemetry stack



There are several approaches to install the telemetry software components. You can download and run each program separately or for example, deploy a chain of Docker containers orchestrated using Docker Compose. This lab leverages this latter approach, except for Pipeline, still launched as separate process (as single binary file).

1. Double click on "Telemetry VM" icon to log on the Ubuntu server that collects the telemetry data.



2. Type `cat ~/environment/docker-compose.yml` to visualize the docker compose configuration file, proposed for this telemetry stack.

```

vagrant@vagrant-ubuntu-trusty-64:~$ cat ~/environment/docker-compose.yml
version: '2'

services:

  # Define an InfluxDB service
  influxdb:
    restart: always
    image: influxdb:1.3
    environment:
      - "INFLUXDB_REPORTING_DISABLED=true"
      - "INFLUXDB_DATA_QUERY_LOG_ENABLED=false"
  
```

```

- "INFLUXDB HTTP LOG ENABLED=false"
- "INFLUXDB CONTINUOUS QUERIES LOG ENABLED=false"
volumes:
- ~/data/influxdb:/var/lib/influxdb
ports:
- "8086:8086"

# Define a service for using the influx CLI tool.
# docker-compose run influxdb-cli
influxdb-cli:
  image: influxdb:1.3
  entrypoint:
    - influx
    - -host
    - influxdb
  links:
    - influxdb

# Define a Kapacitor service
kapacitor:
  restart: always
  image: kapacitor:1.3
  environment:
    KAPACITOR_HOSTNAME: kapacitor
    KAPACITOR INFLUXDB 0 URLS 0: http://influxdb:8086
  volumes:
    - ~/data/kapacitor:/var/lib/kapacitor
    - ~/log/kapacitor:/tmp
  links:
    - influxdb
  ports:
    - "9092:9092"

# Define a service for using the kapacitor CLI tool.
# docker-compose run kapacitor-cli
kapacitor-cli:
  image: kapacitor:1.3
  entrypoint: bash
  environment:
    KAPACITOR URL: http://kapacitor:9092
  volumes:
    - ~/environment/ticks:/root/ticks
  links:
    - kapacitor

grafana:
  restart: always
  image: grafana/grafana:4.5.2
  environment:
    - "GF_INSTALL_PLUGINS=jdbramham-diagram-panel"
    - "GF_SECURITY_ADMIN_PASSWORD=admin"
    - "http proxy=${http proxy}"
    - "https proxy=${https proxy}"
    - "no proxy=environment influxdb 1,${no proxy}"
  ports:
    - "3000:3000"

```

```
links:
  - influxdb
```

NOTE: This configuration file instructs docker-compose to start three main containers (InfluxDB, Grafana and Kapacitor) and two support containers (InfluxDB-CLI and Kapacitor-CLI, that we will use for CLI interactive sessions).

The specified container images and version will be downloaded automatically, if it hasn't been already cached in the system (as in this lab to speed up the installation).

Notice that the file specifies the container specific options, such as:

- **environment** variables to personalize the container, specify log behavior and load plugins (for example, with Grafana, this is the section you should update if you need new plugin, i.e. GF_INSTALL_PLUGINS= jdbraham-diagram-panel example).
- **ports** to map the container network ports to the host system (i.e. TCP port 3000 used by Grafana).
- **volumes** to map files and directory between a container and its host system (i.e. InfluxDB database and logs).
- **links** to stitch containers together using their name and the docker dynamic DNS.

If you need to personalize this environment, update this file before starting the containers in the next step.

3. Type `./telemetry_utility.sh` in the Telemetry VM SSH shell to launch the support script which manages this environment.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ./telemetry_utility.sh

1) Verify Environment
2) Destroy & Clean Environment
3) Start/Stop Pipeline
4) Re-start Pipeline
5) Start/Stop Influx, Grafana & Kapacitor
6) Start/Stop Kafka
7) Toggle Pipeline dump logging
8) Truncate Pipeline dump logging
9) Exit
Please enter your choice:
```

NOTE: As explained in a previous section, this utility calls an ansible-playbooks to instantiate a desired configuration state. For more details, explore the shell scripts with `cat telemetry_utility.sh`.

4. Type 5 (Start/Stop Influx, Grafana & Kapacitor) at the proposed menu, to start the staging of the telemetry consumption stack. Use vagrant as vault password, when queried by the playbook.

```
Please enter your choice: 5

!!! Ready to START InfluxDB, Kapacitor and Grafana !!!

Press any key to continue or Ctrl+C to exit...

Vault password:

PLAY [server]
*****
```



```

TASK [Retriving secrets]
*****
ok: [server]

TASK [Pre-staging phase - including configuration variables]
*****
ok: [server]

TASK [Checking Pipeline process status]
*****
ok: [server]

TASK [Create data directory]
*****
ok: [server]

TASK [Create log directory]
*****
ok: [server]

TASK [Start Influx, Grafana and Kapacitor containers]
*****
changed: [server]

TASK [Pause 10 seconds to let Influx and Grafana to start and download the plugin before configuring]
*****
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [server]

TASK [Creating mdt_db database]
*****
ok: [server]

TASK [Check if InfluxDB datasource exist Grafana]
*****
ok: [server]

TASK [Adding the InfluxDB datasource in Grafana]
*****
ok: [server]

TASK [Update pipeline.conf for Influx]
*****
changed: [server]

TASK [Reload Pipeline (init)]
*****
changed: [server]

PLAY RECAP
*****
server                : ok=12   changed=3    unreachable=0    failed=0

```

NOTE: Review the tasks reported in the playbooks output, for a high-level understanding of the process or explore the actual playbook (i.e. `cat ~/environment/ansible/start influx grafana kapacitor.yml`) for details of the Ansible modules being used and their parameters.

Ansible playbooks are easy to create, self-documenting recipes that define an infrastructure state. They are inefficient for programming logic, that is better to push it in the program calling the playbook (i.e. the bash script `telemetry_utility.yml` in this environment).

5. Type 1 (Verify Environment) at the proposed `telemetry_utility.yml` menu, to verify the status of the environment (docker containers status and access to their exposed API).

```
Please enter your choice: 1

!!! Currently checking for Pipeline, Grafana and Influxdb !!!

Checking Pipeline ... running

    Input Stage: TCP Dial-Out
    Input Stage: gRPC Dial-Out
    Output Stage: TAP dump.txt
    Output Stage: Influx TSDB

Checking Grafana ... running

Checking Influxdb ... running

    Pinging Influxdb API ... OK (204)

Checking Kapacitor ... running

    Pinging Kapacitor API ... OK (204)

Checking Kafka ... down

Checking Zookeeper ... down
```

6. Type 9 (Exit) in the `telemetry_utility.yml` menu to exit the utility.

```
Please enter your choice: 9
BYE
```

7. Issue `grep 'Influx Config' -A 12 ~/environment/pipeline.conf`, to check the InfluxDB output stage appended to the Pipeline configuration file.

```
vagrant@vagrant-ubuntu-trusty-64:~$ grep 'Influx Config' -A 12 ~/environment/pipeline.conf
#<!-- BEGIN Ansible Managed - Influx Config -->
[influx]
stage = xport output
type = metrics
file = environment/metrics.json
datachanneldepth = 1000
#dump = metricsdump.txt
output = influx
influx = http://127.0.0.1:8086
database = mdt db
workers = 10
username=admin
```

```
password=JFfcpyrYibhctUKzdXxN6xR7MpGC20vn+NhfoO/kI0hwhFvYX74pEkfxzyWfWd49IC8tM6z7GI4QNXetTEh1WQRaUgtvk1
Eldb630dBrTGRfCNPbCh4igpdBup3IOfX5t7RpfImcpfIJ0stHY8E5QSddU3yCo+hS1evI/mni8IKmINfHosVHGa9jjQN+v1rbKjBDL
IIyBpKkdLJgTWlso5wyc3bPVEmpMW8QE1FtpcfE5QJC4enabJIACY+e0JtDNbkJ/bWnCW58dMLjOdRz9W7Yw1ASmO2XfHfZt9ddvdGe
h3oCL3yAtoj6CiFoPAJzoKQa8D7KFRCCjdQj807IEM95EMqrY0g+9fESsCWXQ3/JEZqrz1V/Uq2sepyDI5ItOQ90eet76FEdLDCuOGk6
avx18O6fpNiFzk/IRlGehq9ruUQRVxvhoG8Aq4ZQH6CNh//AN+QwTF+Pb8qZpX82v35SmO5ESAR1vxaQn17TNeUWg7ulSLKqnoBicRX9
Jw3vsA67vcdmVsm+PcZmgnYlk3QDIHp6u8VBPopYx7ZU76/i3F27dbY92/rQdCV1ICbZhAXnRda8BCeYoqCjnO2PfEEODoWP6wgOHwj
MsFSA4qVEX5Eh4AvglD6wbMynw27kZozOyM0+CqUJ3XlK1EEpeKvFoLCvyO2dXXKHj8w=
#<!-- END Ansible Managed - Influx Config -->
```

NOTE: Notice that as already explained in the gRPC section, we are using the Pipeline PEM file to encrypt the InfluxDB password (which is `admin`) to avoid being challenged to type the password, when Pipeline connects to InfluxDB.

8. Issue `docker ps` command to check manually the status of the docker containers running on the system.

```
vagrant@vagrant-ubuntu-trusty-64:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
6c291c128c59	kapacitor:1.3	"/entrypoint.sh kapa..."	33 minutes ago	Up 33 minutes
e8f5822e5e9f	grafana/grafana:4.5.2	"/run.sh"	33 minutes ago	Up 33 minutes
8fec8c9d9fc4	influxdb:1.3	"/entrypoint.sh infl..."	33 minutes ago	Up 33 minutes

NOTE: Three out of the five requested containers are active. InfluxDB-CLI and Kapacitor-CLI have already exited and they will be launch again in later sections, when we need an interactive CLI session toward Influx or Kapacitor.

Section 5: Understanding InfluxDB

"InfluxDB is a Time Series Database built from the ground up to handle high write & query loads. InfluxDB is a custom high performance datastore written specifically for timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytics. Conserve space on your machine by configuring InfluxDB to keep data for a defined length of time, and automatically expiring and deleting any unwanted data from the system. InfluxDB also offers a SQL-like query language for interacting with data" <https://www.influxdata.com/time-series-platform/#influxdb>.

InfluxDB stores timeseries that are sequences of data points as successive measurements from the same source over an interval of time. Each data point has one or more keys (indexes that identifying a source), one or more fields (storing the actual metric data) and a timestamp.

This section documents the basic command to explore the InfluxDB, where Pipeline is exporting the streaming telemetry metrics that we have configured in the first part of this lab. This section explores how to use InfluxDB CLI and the next section shows how to use InfluxDB API, to retrieve the same information.

For more information and video tutorial on InfluxDB and InfluxSQ, visit <https://www.influxdata.com/university>.

Steps

1. Type `cd ~/environment && docker-compose run influxdb-cli` to run an interactive container for the InfluxDB CLI.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cd ~/environment && docker-compose run influxdb-cli
WARNING: The http_proxy variable is not set. Defaulting to a blank string.
WARNING: The https_proxy variable is not set. Defaulting to a blank string.
WARNING: The no_proxy variable is not set. Defaulting to a blank string.
Starting environment_influxdb_1 ... done
Connected to http://influxdb:8086 version 1.3.7
InfluxDB shell version: 1.3.7
>
```

2. Type the command `show databases` to visualize the available databases.

```
> show databases
name: databases
name
----
mdt_db
_internal
```

NOTE: if you check [Pipeline configuration file](#), `mdt_db` was specified as database to store the streaming telemetry data.

3. Type the command `use mdt_db` to select the `mdt_db` as working database.

```
> use mdt_db
Using database mdt_db
```

4. Type the command `show measurements` to retrieve the list of measurements in the database.

```
> show measurements
name: measurements
name
```

```

----
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters
Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-rib-route-table-
name/protocol/bgp/as/information
Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary
Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization

```

NOTE: A measurement describes what has been stored in the database and in our use case, it provides a one to one relationship with the sensor-paths we have configured on the routers.

5. Type `show tag keys` from "`<measurement_name>`" for a list of tags or indexes selected for a particular measurement. Choose as "`<measurement_name>`" any of the available measurements retrieved in step 4 or use the `data-rate` path, as in the example below, to follow the rest of the document examples.

```

> show tag keys from "Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate"
name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
tagKey
-----
EncodingPath
Producer
interface-name

```

NOTE: Remember to add double quotes (") for measurements, tags or fields that include the character desh/minus sign (-). The correct use of quote is subject of some frustrating experiences when using InfluxQL.

6. Type `show tag values` from "`<measurement_name>`" with `key = "<tag_name>"` to retrieve the actual values stored for a particular key, by substituting `<measurement_name>` and "`<tag_name>`" with any combination of metrics and tag names retrieved in step 4 and 5.

```

> show tag values from "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate" with key = "interface-name"

name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
key          value
---          -
interface-name GigabitEthernet0/0/0/0
interface-name GigabitEthernet0/0/0/1
interface-name GigabitEthernet0/0/0/2
interface-name MgmtEth0/RP0/CPU0/0
interface-name Null0

```

7. Type `show series` from "`<measurement_name>`" for a list of the timeseries for a specific measurement. InfluxDB stores timeseries as sequences of data points from the same source. A source in this context, is a combination of the same tag values in a specific measurement.

```

> show series from "Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate"
key
---
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,EncodingPath=Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,Producer=test XR,interface-name=GigabitEthernet0/0/0/0

```

```

Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,EncodingPath=Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,Producer=test_XR,interface-name=GigabitEthernet0/0/0/1

Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,EncodingPath=Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,Producer=test_XR,interface-name=GigabitEthernet0/0/0/2

Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,EncodingPath=Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,Producer=test_XR,interface-name=MgmtEth0/RP0/CPU0/0

Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,EncodingPath=Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-
rate,Producer=test_XR,interface-name=Null0
>

```

NOTE: In this data-rate example, we found at step 5 that it has three tag keys (EncodingPath, Producer and interface-name). The combination of tag keys will generate a different time series, one for each of the five interfaces from that same test router.

8. Type `show field keys` from "`<measurement_name>`" for a list of fields exported for a particular measurement and their types. Choose as "`<measurement_name>`" any of the available measurements retrieved in step 4.

```

> show field keys from "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate"
name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
fieldKey      fieldType
-----
bandwidth      integer
input-data-rate float
input-load      integer
input-packet-rate float
load-interval   integer
output-data-rate float
output-load      integer
output-packet-rate float
peak-input-data-rate float
peak-input-packet-rate float
peak-output-data-rate float
peak-output-packet-rate float
reliability     integer

```

9. With the information retrieved in the previous steps, we are ready to explore a basic InfluxQL query: `SELECT <a list of field keys (use * for all of them)> FROM <a measurement> WHERE <matching a list of tag names = values> LIMIT <desired number of results>`.

Execute the query in bold in the following example or compose one of your own with the fields, measurements and keys retrieved in the previous steps.

NOTE: When exploring InfluxQL, remember to add the final option `LIMIT` to control the number of data point retrieved and avoid having to wait for a long list. Also, be careful to use single quote (') for the tag string values because InfluxDB is quite picky on this and it reject double quotes (").

```

> SELECT "input-data-rate", "output-data-rate" FROM "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate" WHERE Producer='test_XR' AND "interface-name" =
'GigabitEthernet0/0/0/0' LIMIT 5
name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate

```

time	input-data-rate	output-data-rate
----	-----	-----
1512702331500000000	10	344
1512702332542000000	9	338
1512702333588000000	9	338
1512702334626000000	9	338
1512702335673000000	9	339

10. Type `precision rfc3339` to change InfluxDB default timestamp format with the time and date commonly used in the Internet.

```
> precision rfc3339
```

NOTE: By default, InfluxDB reports timestamps as epoch time: the amount of time in nanoseconds that has elapsed since 00:00:00 UTC, Thursday 1 January 1070.

11. Execute again the query from step 9 to observe the change in the time format.

```
> SELECT "input-data-rate", "output-data-rate" FROM "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate" WHERE Producer='test_XR' AND "interface-name" =
'GigabitEthernet0/0/0/0' LIMIT 5
name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
time                input-data-rate output-data-rate
----                -
2017-12-08T03:05:31.5Z  10          344
2017-12-08T03:05:32.542Z 9          338
2017-12-08T03:05:33.588Z 9          338
2017-12-08T03:05:34.626Z 9          338
2017-12-08T03:05:35.673Z 9          339
```

NOTE: remember that the timestamp is expressed in UTC time (the z at the end of the time identifies UTC).

12. (Optional) The previous examples explain the most basic query concepts in InfluxDB. Copy and execute the following example queries (or adjust these examples with your own with the fields, measurements and keys retrieved in the previous steps) to answer some of the most common queries, like:
- the last (from a timestamp prospective) element in a time series
 - the mean of a time series
 - the list of data points for a time series in the last 1 minute

```
> SELECT last("input-data-rate") FROM "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate" WHERE Producer='test_XR' AND "interface-name" =
'GigabitEthernet0/0/0/0'
name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
time                last
----                -
2017-12-11T00:50:05.694Z  10
>
> SELECT mean("input-data-rate") FROM "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate" WHERE Producer='test_XR' AND "interface-name" =
'GigabitEthernet0/0/0/0'
name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
time                mean
----                -
1970-01-01T00:00:00Z  9.269338067461323
```

```
>
> SELECT "input-data-rate", "output-data-rate" FROM "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate" WHERE Producer='test_XR' AND "interface-name" =
'GigabitEthernet0/0/0/0' AND time > now() - 1m

name: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate
time                input-data-rate output-data-rate
----
2017-12-11T00:56:14.757Z 10          378
2017-12-11T00:56:15.798Z 10          378
2017-12-11T00:56:16.902Z 10          378
2017-12-11T00:56:18.047Z 10          378
2017-12-11T00:56:19.071Z 10          378
2017-12-11T00:56:20.114Z 10          378
<SNIP>
```

NOTE: Consider to check https://docs.influxdata.com/influxdb/v1.3/query_language/functions, if you are interested in more timeseries purpose built functions supported by InfluxDB or follow the InfluxQL training available at <https://www.influxdata.com/university>.

13. As final concept for this introductory section, type `show retention policies` to visualize for how long the `mdt_db` has been configured to store the telemetry data points, before making space for new data and the number of replicas.

```
> show retention policies
name    duration shardGroupDuration replicaN default
----    -
test6h  6h0m0s   1h0m0s             1         true
```

The default InfluxDB, auto-generated retention stores the data points for 7 days with a replica of 1 (this is a nice article if you are interested in more details: <http://www.oznetnerd.com/influxdb-retention-policies-shard-groups>).

As displayed in the example, this environment has been configured to retain the data for six hours, stored in measurement files of one hour each (shard duration).

You can change the `mdt_db` database retention policy by altering the CREATE DATABASE option in

`start_influx_grafana_kapacitor.yml` playbook before creating the Influx environment with the `./telemetry_utility.sh` utility.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat ~/environment/ansible/start_influx_grafana_kapacitor.yml | grep
6h
    body: "q=CREATE DATABASE mdt_db with duration 6h replication 1 shard duration 1h name test6h"
```

14. Type `exit` to close the InfluxDB CLI Docker container.

```
>
> exit
```


Section 6: Exploring InfluxDB APIs

The previous section explored InfluxDB concept using the InfluxDB CLI. This short section builds on these concepts explaining how you can use InfluxDB REST API, to interact with the database. Familiarize with InfluxDB REST API is important, if you plan to run these queries from an application.

1. Copy and paste the highlighted curl command below (as a single line) in the Ubuntu CLI to retrieve the list of databases in InfluxDB. Curl or Client for URLs (cURL) is a useful utility to test your RESTful queries. In this environment InfluxDB listen on TCP port 8086 and answers to queries with the option `q=<query_string>`.

```
vagrant@vagrant-ubuntu-trusty-64:~/environment$ curl -G 'http://localhost:8086/query?pretty=true' --
data-urlencode "q=show databases"
{
  "results": [
    {
      "statement_id": 0,
      "series": [
        {
          "name": "databases",
          "columns": [
            "name"
          ],
          "values": [
            [
              "mdt_db"
            ],
            [
              "_internal"
            ]
          ]
        }
      ]
    }
  ]
}
```

IMPORTANT NOTE: in the example above, we have used localhost to query InfluxDB API from the same system. When spinning this environment outside dCloud (as explained in [section 16](#)), you can substitute this IP address with one of the host external interface IP and query InfluxDB remotely.

2. Copy and paste the cURL command in the example below, to re-execute our [first InfluxSQ select query](#) as API call. You can apply the same logic to the rest of the CLI queries, that have been explored in the previous section.

NOTE: We need a first `--data-urlencode` parameter to specify the db name (`mdt_db` in this example) and that we have to escape all double quote used in the original InfluxQL string with the backslash character (`\`) for a proper syntax.

```
vagrant@vagrant-ubuntu-trusty-64:~/environment$ curl -G 'http://localhost:8086/query?pretty=true' --
data-urlencode "db=mdt_db" --data-urlencode "q=select \"input-data-rate\", \"output-data-rate\" from
\"Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/data-rate\" where
Producer='test_XR' and \"interface-name\" = 'GigabitEthernet0/0/0/0' limit 5"
{
```

```

"results": [
  {
    "statement id": 0,
    "series": [
      {
        "name": "Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate",
        "columns": [
          "time",
          "input-data-rate",
          "output-data-rate"
        ],
        "values": [
          [
            "2017-12-11T03:37:26.414Z",
            10,
            380
          ],
          [
            "2017-12-11T03:37:27.478Z",
            10,
            380
          ],
          [
            "2017-12-11T03:37:28.527Z",
            10,
            380
          ],
          [
            "2017-12-11T03:37:29.609Z",
            10,
            380
          ],
          [
            "2017-12-11T03:37:30.645Z",
            10,
            380
          ]
        ]
      }
    ]
  }
]
}

```

3. Copy and paste the cURL command in the example below, to delete all data series from the InfluxDB, without impacting the rest of the environment.

```

vagrant@vagrant-ubuntu-trusty-64:~/environment$ curl -POST 'http://localhost:8086/query' --data-
urlencode "db=mdt_db" --data-urlencode "q=DROP SERIES FROM *.*"

{"results":[{"statement id":0}]} # this line confirm the success of the operation

```

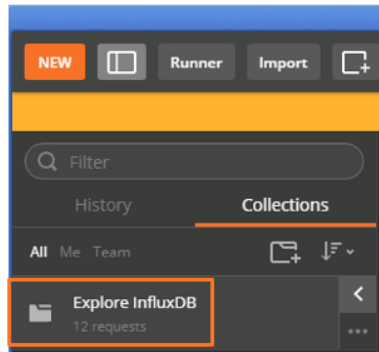
4. Type `exit` to close the SSH session to the Ubuntu host.

- Double click on the Postman icon to launch the program.

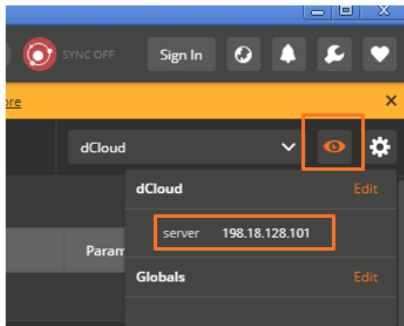


NOTE: Postman is a graphical interface that simplifies managing collection and execution of RESTful commands.

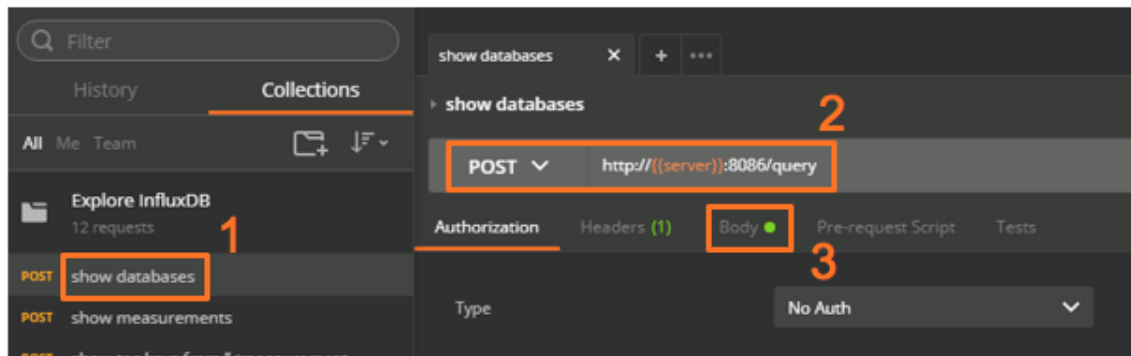
- Click on the `Explore InfluxDB` collection to show the list of pre-populated REST calls.



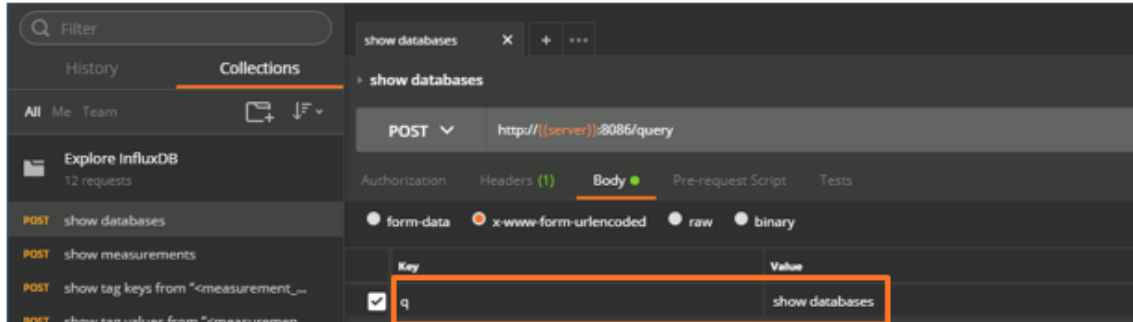
- Click on the eye icon (on the top right) to visualize the variables for the dCloud environment. `{{server}}` variable will be substituted with the IP address hosting InfluxDB when executing any of the following REST commands.



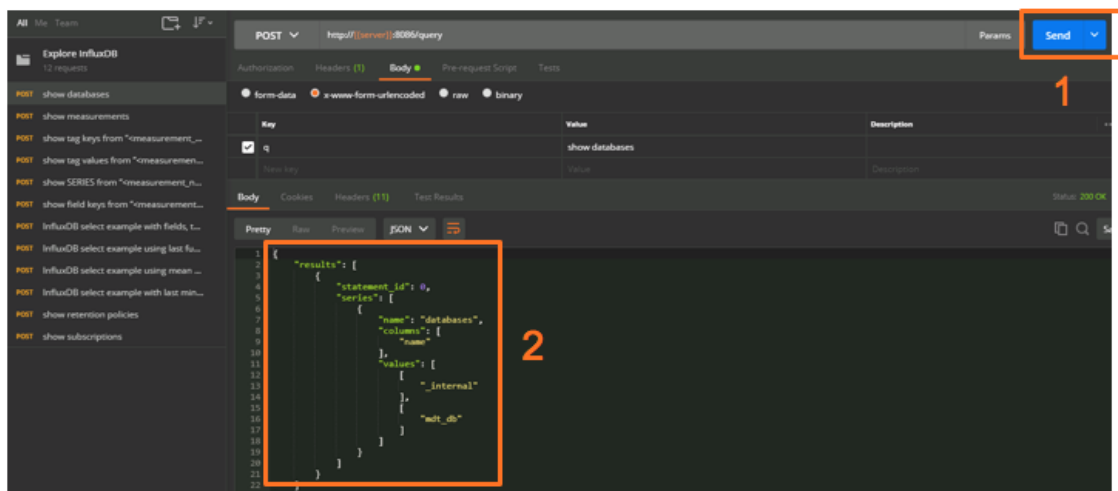
- Click on `show databases` call (1) to visualize its content, validate that the POST query (2) is the same API used in the previous examples and click on Body (3), to check the actual query to be submitted to InfluxDB.



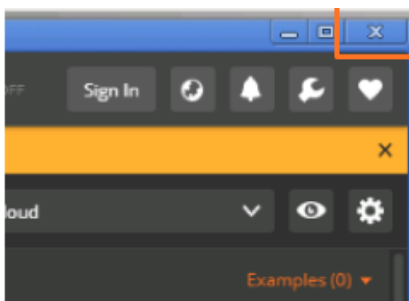
9. Validate that the key/value configured for this query matches our [initial example](#).



10. Click on Send (1) send the query and review the result in button pane (2) that includes our `mdt_db`.



11. You should be already familiar with the rest of the queries, discussed in the previous section. You can decide to explore them (following the same process explained in the previous 3 steps) or close Postman by clicking on the X window button and move the next section.



Section 7: Exploring Grafana - My first dashboard

Grafana allows to query, visualize (i.e. graph, histograms, heatmaps), alert on metrics, stored in different databases (i.e. InfluxDB, Prometheus, Elasticsearch) and create interactive, easy to customize dashboards.

Check <https://grafana.com/grafana> for a snapshot of Grafana capabilities or <http://play.grafana.org/dashboard/db/grafana-play-home?orgId=1> to review some working demo dashboards.

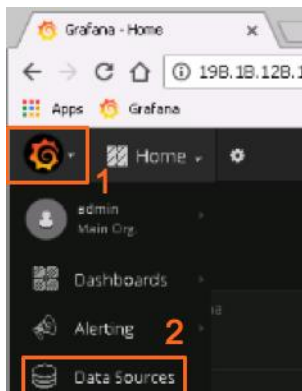
1. Double click on the Google Chrome icon to launch the browser.



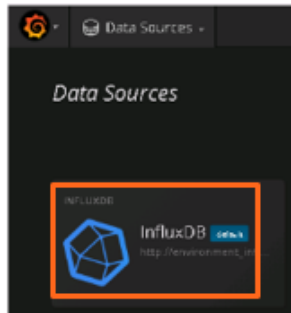
2. Chrome is configured to open Grafana home page (listening `http://198.18.128.101:3000`) or click on Grafana bookmark (1). Type the username `admin` (2), the password `admin` (3) and click on the `Log in` button (4).



3. Click on the Grafana icon (1) and on the Data Source (2) in the proposed dropdown list.

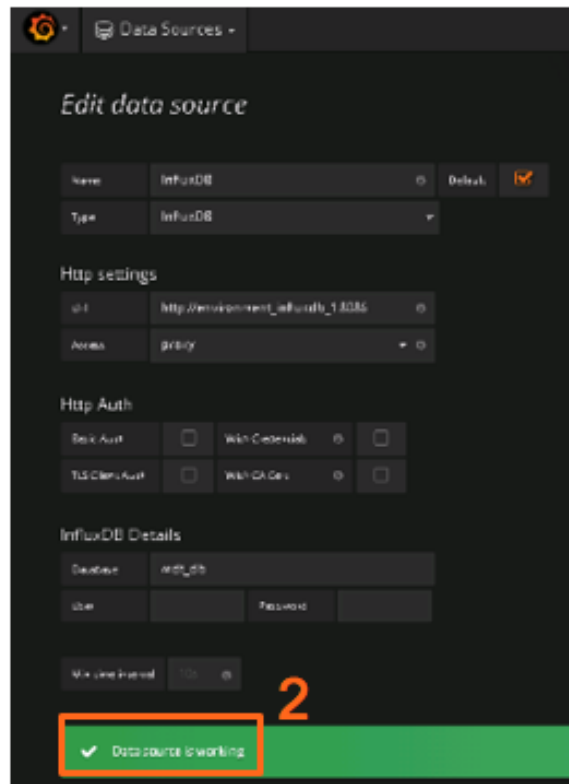
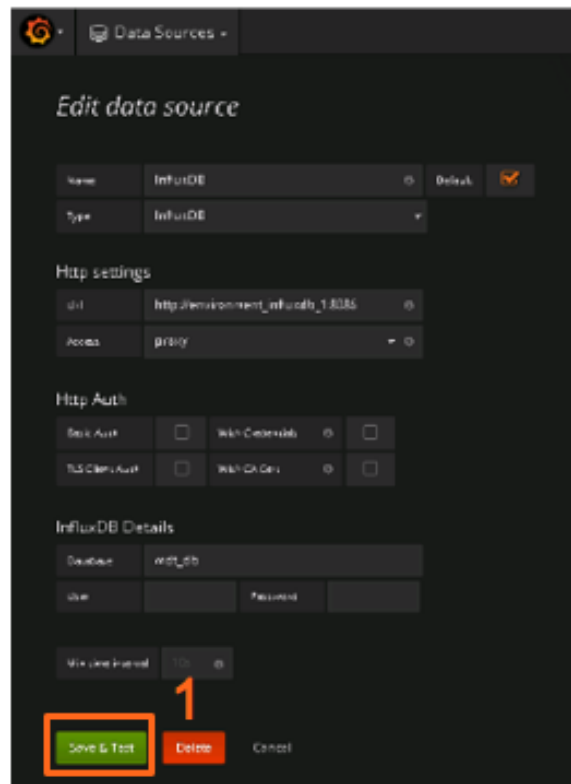


4. Click on the InfluxDB data source to visualize its details.

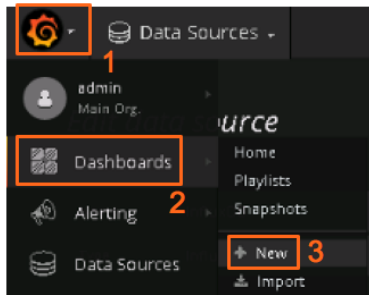


NOTE: The Grafana data store describes how Grafana interacts with the timeseries database, in this case the InfluxDB database. This data store was configured when executing `~/environment/ansible/start_influx_grafana_kapacitor.yml` playbook. Review this file if you are interested in the specific REST call.

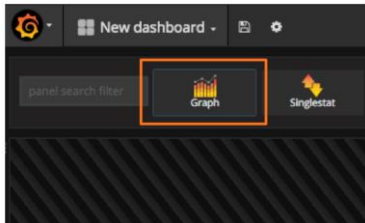
- Review the data source details, in particular the URL of the Influx database (listening on `http://198.18.128.101:8086`), the proxy mode (requesting Grafana backend to run the queries toward InfluxDB) and the `mdt_db` database (already discussed in the previous section). Click on the Save and Test button (1) to confirm the connectivity between Grafana and InfluxDB and confirm the successful message (2). Remember this procedure to validate Grafana data source, when you experience issues in your dashboards.



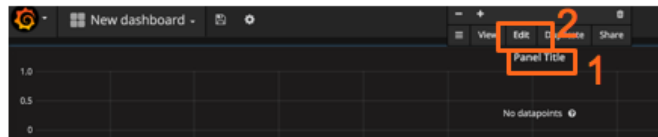
6. Create a new dashboard by clicking the Grafana icon and choose Dashboards > + New.



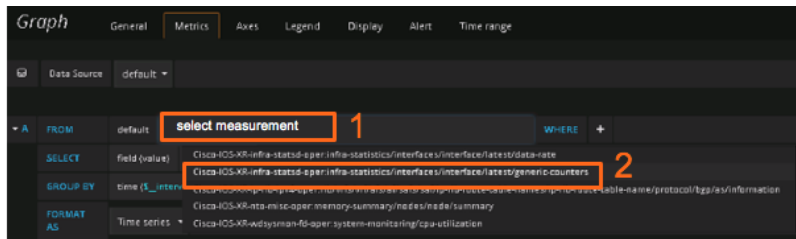
7. Add a graph by clicking on the Graph icon.



8. Click on Panel Title (1) and then Edit (2) to update the default empty graph.

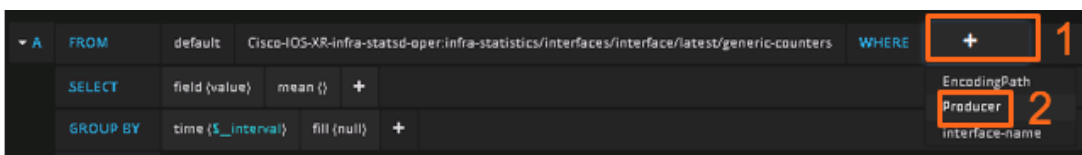


12. Click on `select measurement` (1) and choose `Cisco-IOS-XR-infra-stats-oper:infra-statistics/interfaces/interface/latest/data-rate` measurement in the proposed list (2).

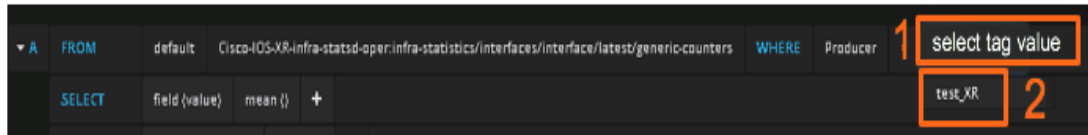


NOTE: the list of measurements (like the rest of the selectable fields) are the result of a real time Grafana queries toward InfluxDB.

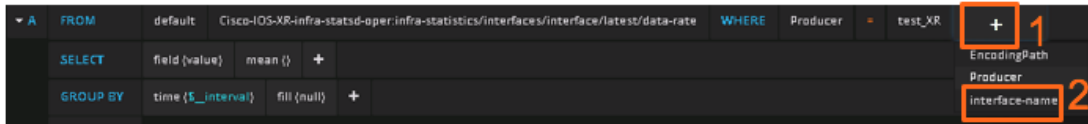
13. Click on + sign (1) and choose `Producer` in the proposed list (2), to filter by the router producing the measurements.



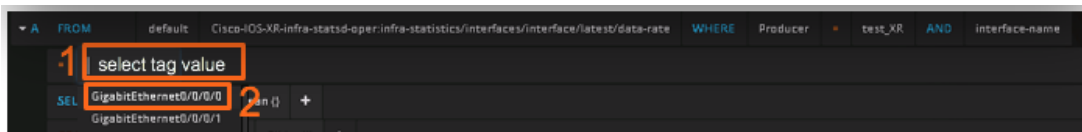
14. Click `select tag value` (1) and choose `test_XR` that is our test router.



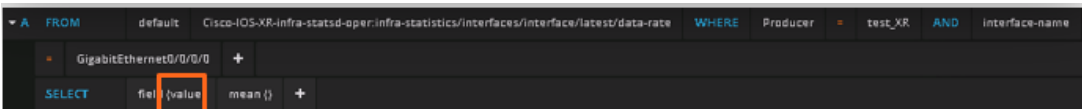
15. Click on + sign again (1) and choose `interface-name` in the proposed list (2), to filter by a specific interface.



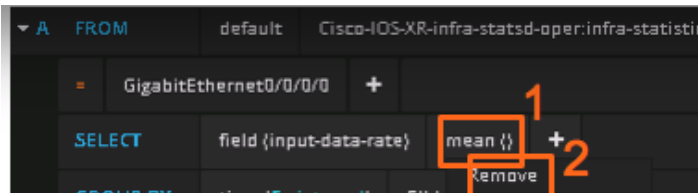
16. Click on `select tag value` (1) and choose for example, `GigabitEthernet0/0/0/0`.



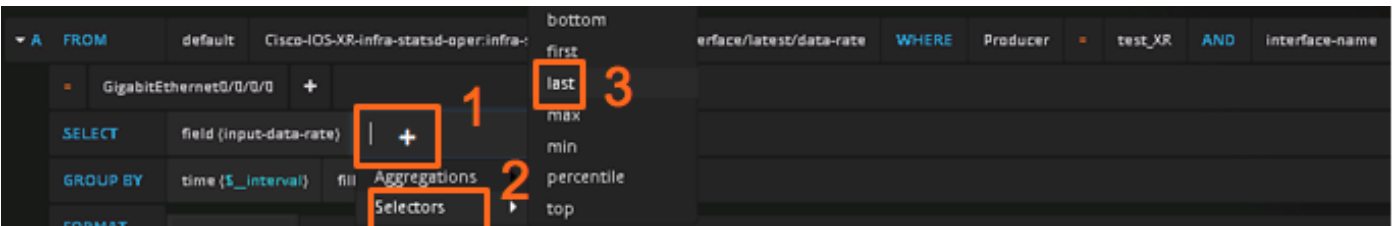
17. Click on the field `value` and choose for example `input-data-rate` in the list of fields for this specific measurement.



18. Click `mean` field (1) and choose `Remove` (2).

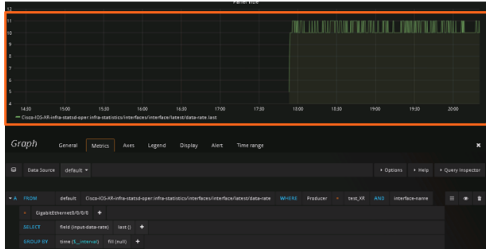


19. Click on + sign after the field value (1), choose `Selectors` in the proposed list (2), and `last` (3) in the options of Selectors.

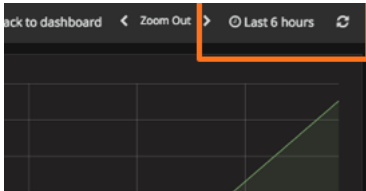


NOTE: The step here suggests configure Grafana to retrieve and graph the last metric value, instead of computing the mean. Using the same procedure, you can configure the rest of time series specific functions, implemented natively in InfluxDB.

20. Check the top section of the window, you should see graph for the query that we have finishing composing.



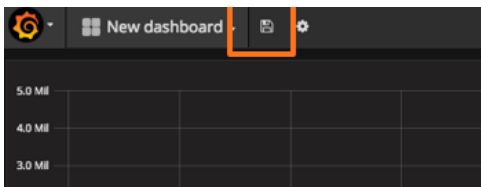
21. By default, a new graph visualizes the last 6 hours of traffic and it doesn't refresh the data. Click on the **Last 6 hours** on the top right to access the time setup.



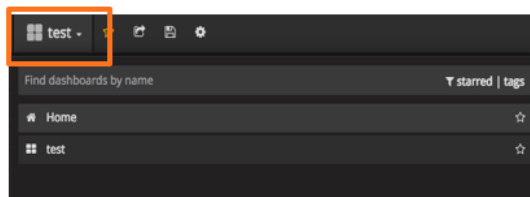
22. You can now select in section 1 a specify time range (as "From/To" format), in section 2 a refresh frequency (5 seconds which is the minimum for interactive dashboard) and apply by clicking on Apply. Alternatively, you can directly select in section 4 a quick time. Select for example 5 second refresh, apply and validate that your initial graph starts refreshing.



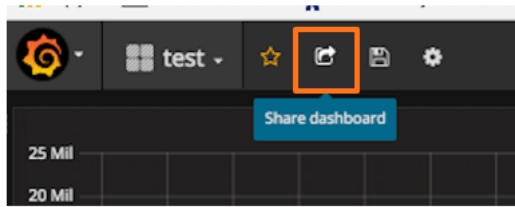
23. Save the dashboard by clicking on the disk icon, providing a name of your choice (in this example "test") and clicking on Save.



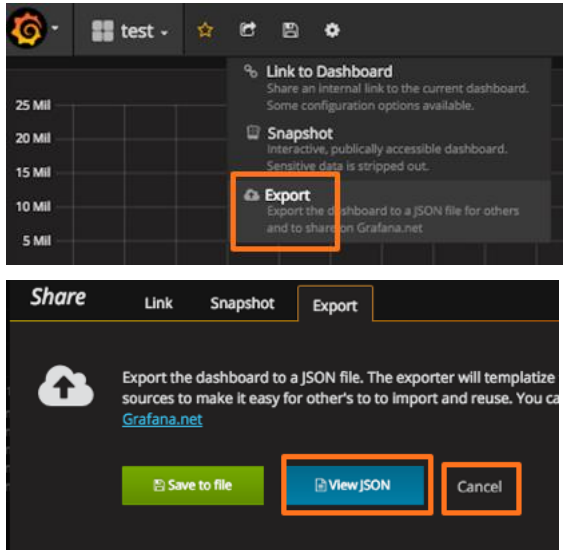
16. our dashboard has been added to the dashboard list and you can re-access it by clicking on the dashboard list and selecting the specified name.



17. Click on **Share dashboard** icon.



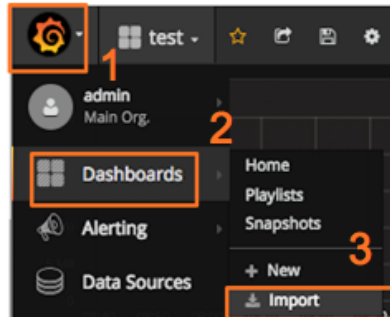
18. Select **Export** to access the option of saving the dashboard definition on a file or visualizing as JSON encoding (that you can copy and paste). Chose **View JSON** and after viewing the output, close the JSON details tab and click on cancel.



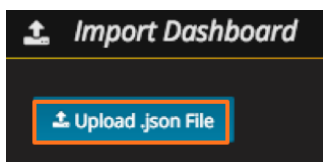
Section 8: Exploring the environment reference dashboard

In this section, we are going to build on the Grafana concepts just learned by importing and exploring a new dashboard that reports on common day to day network metrics (i.e. CPU, memory utilization and interfaces utilization or BGP routes) from a traditional vs near-real time prospective.

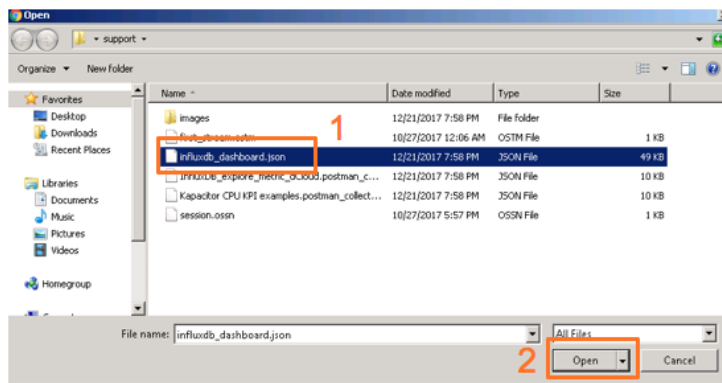
1. Clicking the Grafana icon (1) and choose Dashboards (2) > + Import (3) to import a predefined dashboard that you can use to customize your POCs. You should follow this same process to re-import a dashboard exported as JSON file in the previous step.



2. Click on Upload .json File .

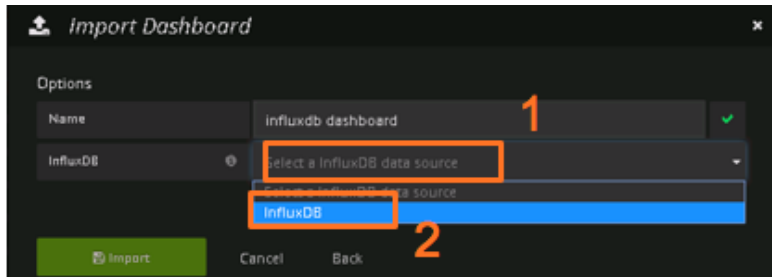


3. Click `Influxdb_dashboard.json` (1) to select the file and on Open button (2).

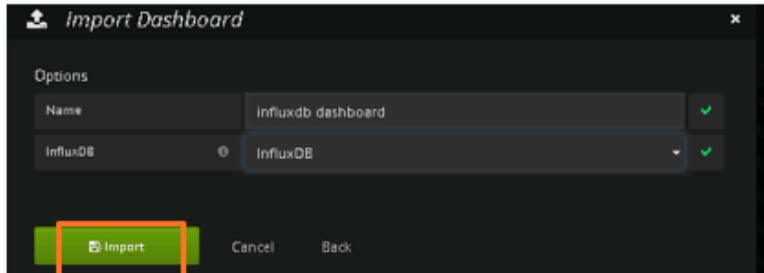


NOTE: `Influx_dashboard.json` is stored in `Desktop/support` directory, navigate to this directory, if the window proposes a different path.

4. Click on Select an Influxdb data source (1) and select InfluxDB from the proposed list (2).



5. Click on the `Import` button to complete importing the dashboard.

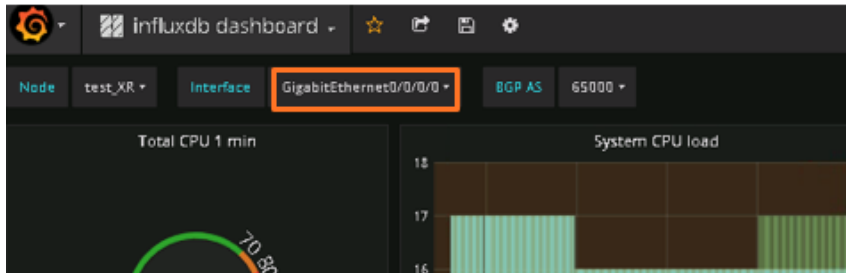


6. You should now have access to the proposed environment reference dashboard, that should be already rendering the model driven information streamed from the XRv test router. Explore the default project's dashboard, it provides common metrics (i.e. CPU, memory, routing information and processes details) that collected today from the network infrastructure using computationally expensive SNMP queries and CLI scraping techniques.

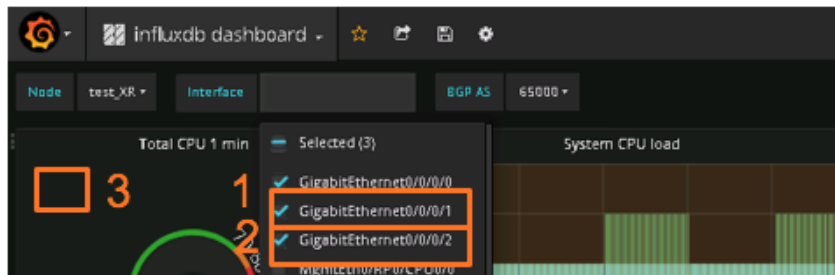


Note: the following steps assume the `test_XR` previously configured is still streaming. If you are starting model driven telemetry on the router after loading this dashboard, reload this page because the dashboard has been configured to populate the variable fields like `Node`, `interface` and `BGP_AS` when loading (more on these templates later in the section).

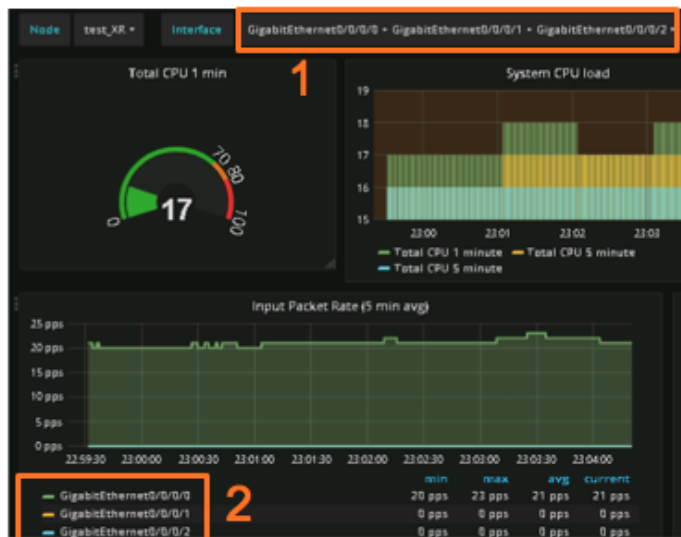
7. Click on the `GigabitEthernet0/0/0/0` field to visualize the list of interfaces.



8. Click on GigabitEthernet0/0/0/1 (1) and GigabitEthernet0/0/0/2 (2) check boxes to selected these interfaces and finally, click on an empty space (3) in the dashboard to confirm this choice.

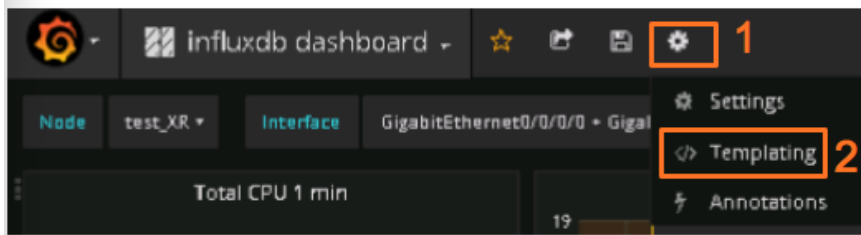


9. Confirm that the dashboard is now reporting metrics for these newly added interfaces.

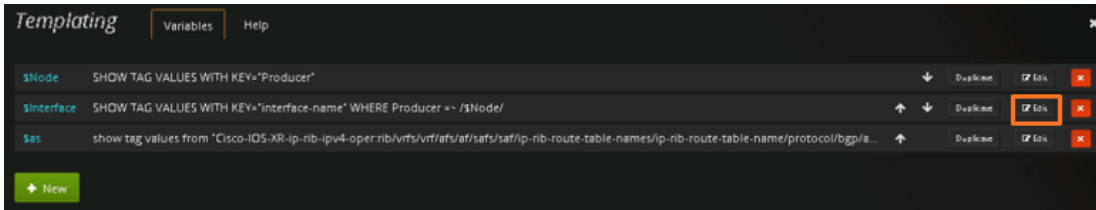


NOTE: Node, Interface and BGP_AS selection fields provide options to dynamically change the details represented in the dashboard and they are created using Grafana's templates (see next steps).

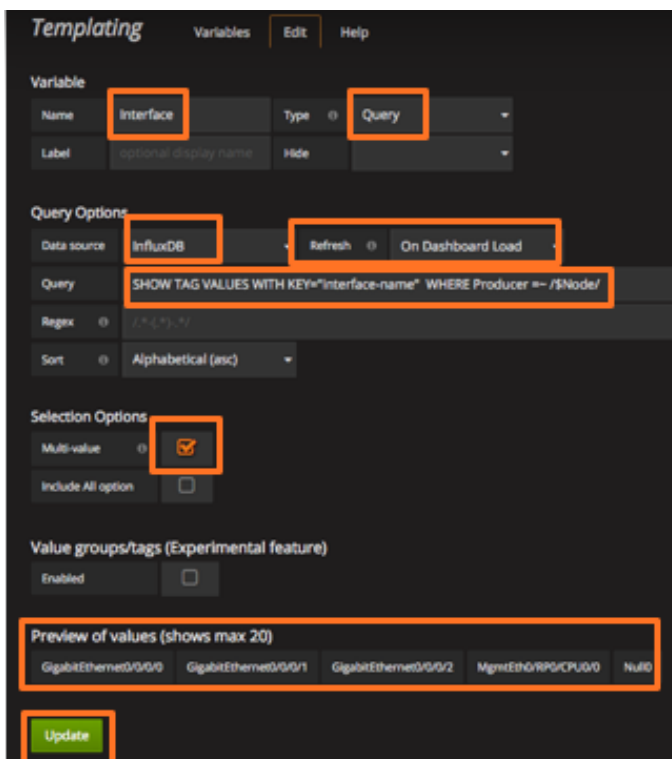
10. Click on the dashboard setting wheel and select `Templating` in the list of options.



11. As expected there are three templates (\$Node, \$Interface and \$as) because we have three dynamic fields in the dashboard. Click on the \$Interface Edit option to visualize its details.



12. Check the fields required to configure a dynamic template, that retrieves the data as a InfluxQL query, using the InfluxDB data source. Click on the Update button, when you finish investigating the template.



NOTE: Notice that you can choose between Grafana populating the template fields when the dashboard page is loaded (as used in this example) or at time change (which may be more resource intensive at 5 seconds refresh). Also notice that the template provides a preview for the values returned by the query that will be used to populate the dashboard.

13. Click on the X to close the Templating view.



NOTE: Grafana support a lot of visualization options and plugins to extend the default list. A nice website to explore these options is <http://play.grafana.org/dashboard/db/grafana-play-home?orgId=1> that provides a test live environment.

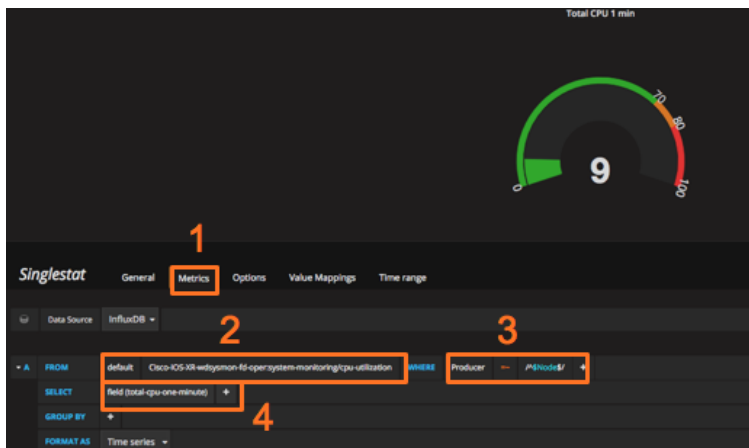
In the previous section, we have created a basic graph, by selecting the **Graph** option. In the next steps, we will explore how a Singlestat (in particular a Gauge) is configured.



14. Click on the **Total CPU 1 min** graph title (1), followed by a click on **Edit** (2) in the proposed menu, to access the graph's configuration details.

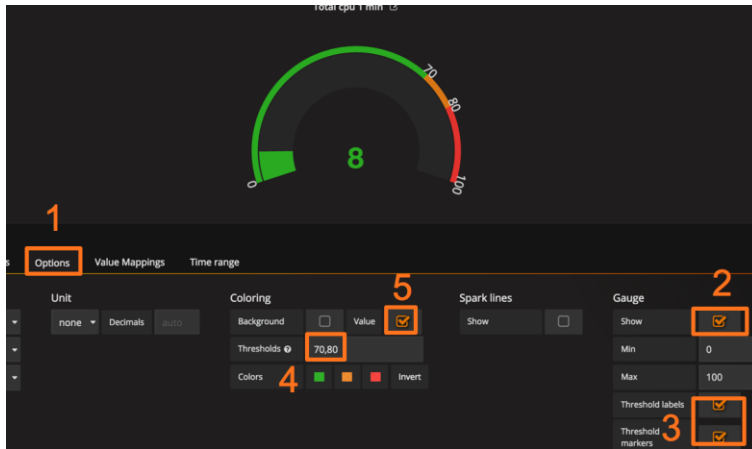


15. Confirm that you are presented with the **Metrics** view (1), that defines the graph query details. As learned in the previous section, the FROM field specifies the measurement (2) followed by the WHERE (3) and SELECT (4) options.

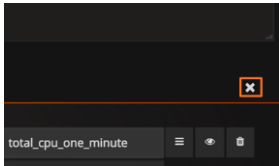


NOTE: Instead of specifying a hardcoded Producer (3), this graph uses the `$Node` template, to dynamically populate this field, based on the selected variable. Click on `/^$Node$/` to confirm that Grafana prepopulates as an option, the value fields with the list of Templates and the actual values retrieved from querying the data source.

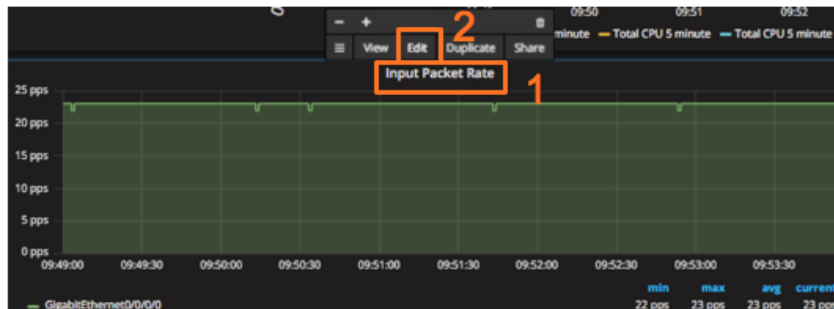
16. Click on the **Options** view (1) and explore the available configuration: disable and re-enable the gauge by clicking on **Show** (2), disable and re-enable the threshold options (3), update the proposed thresholds (4) and de-select, select the visualization of the threshold color on the actual value (5). Check how the gauge changes, based on your actions.



17. Click on the X sign to exit the graph detail view.

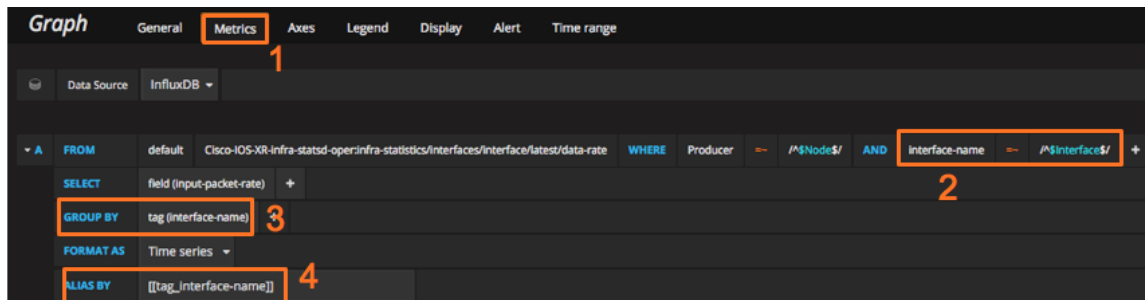


18. Click on `Input Packet Rate` graph title (1) to access its option menu and select Edit (2). This is an example of Graph, as we created in our initial Dashboard.



NOTE: Input/Output Packet Rate and Data Rate are common interfaces metrics that we collect today using SNMP or CLI scraping. The router interface has been configured with the common 'load-interval 30' to minimize the sliding average computation interval to 30 seconds but remember that we are still pushing the metric every 1 second (much faster than what we are used to collect with traditional technologies). We can consider this example as the best case of using the current extraction methods, when we compare with the other near real-time examples.

19. Click on the Metrics view (1) to access the query details.

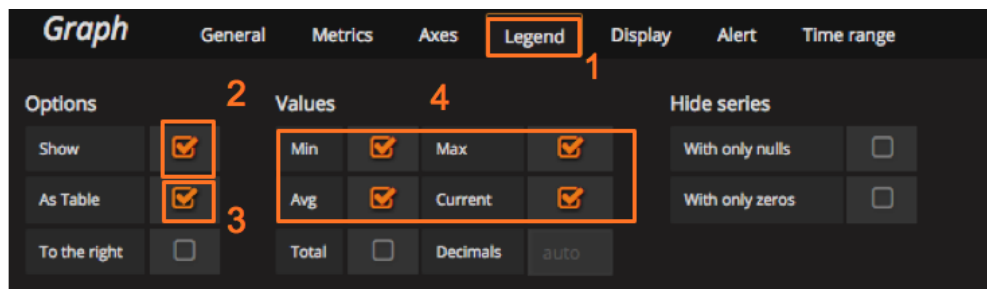


NOTE: You should be already familiar with the meaning of FROM, WHERE and SELECT from the previous steps, let's explore how to use the interface-name template to dynamically render multiple graphs from a single query. In the Gauge example, we explored the meaning of /\$Node\$/ for specifying the provider, we can now use the same mechanism, but with \$interface template, to specify in the WHERE clause (2) the interfaces to render.

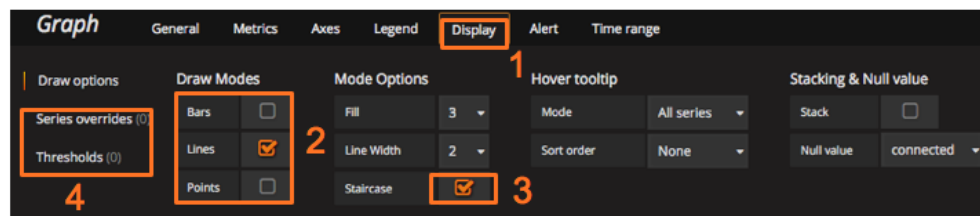
\$interface template has been configured as multi-selection field, allowing to select multiple interfaces and we must remember to specify the GROUP BY (3) field to render a graph per interface.

Finally, we can use Grafana Alias pattern (4) to dynamically render the interface name (see more for dynamic alias patterns <http://docs.grafana.org/features/datasources/influxdb>).

20. Click on Legend (1) to access the graph's legend options. Select and deselect the Show (2) to add and remove the legend, As Table (3) to visualize not only the interfaces names but also some associated values and Values (4) to choose which values to report on.

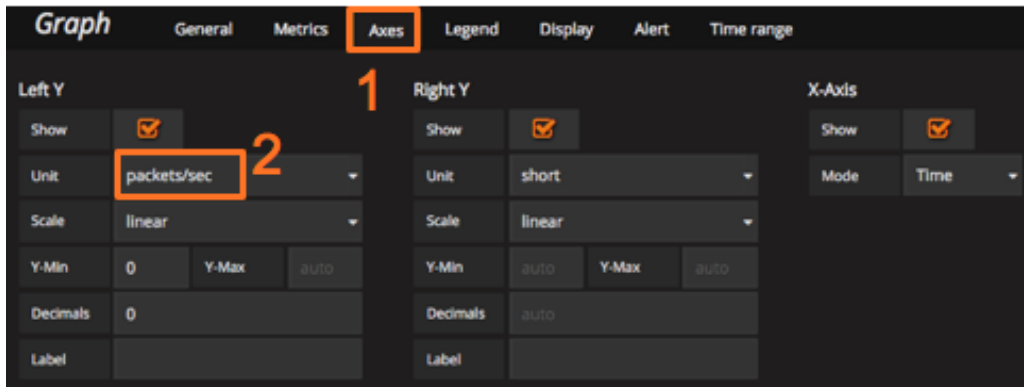


21. Click on Display to access the graph personalization options. Select and de-select the Bars, Lines, Points (2) and Staircase (3) options to understand how this change the graph. For example, this reference dashboard uses Lines for the interface statistics but Bars are used for CPU and memory utilizations.

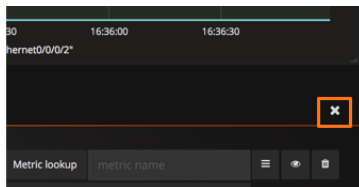


NOTE: Series overrides (4) is a more advanced Grafana feature not explored in this Lab. It is an interesting functionality because it can render different queries on the same graph. For example, show the average min and max sliding deviation range as baseline, and impose the actual measurement. Thresholds supports triggering basic alerts but it is not explained in this lab because it cannot be used when leveraging templating in the queries.

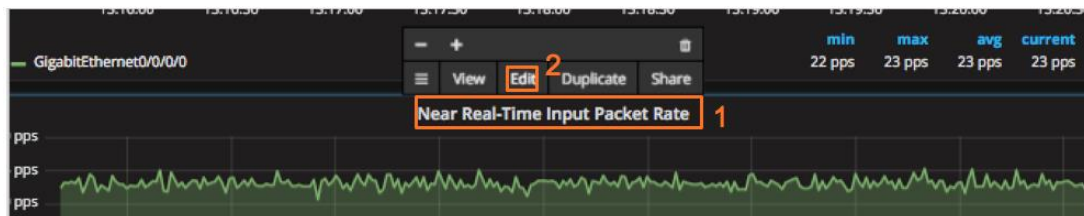
22. Finally, Click on Axes (1) to access the axes personalization window and explore the different options (2) to specify the measurement units (packet/sec in this example).



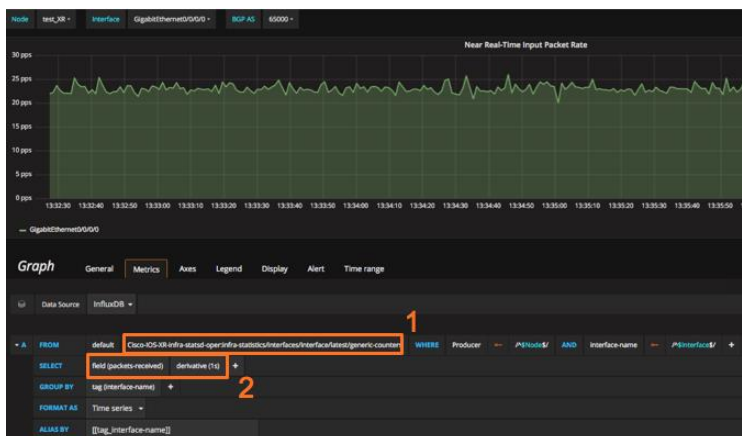
24. Click on the X sign to exit the graph detail view.



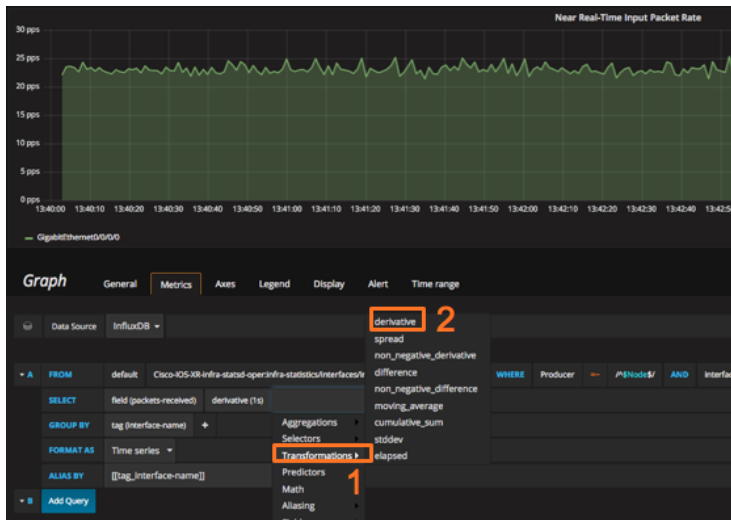
25. Click on Near Real-Time Input Packet Rate (1) and select Edit (2), in the proposed menu, to visualize the details for computing measurements using InfluxDB native functions. More specifically, this example calculates the near real-time packet rate as derivative of the packet count.



26. Check the measurement specified in the FROM field (1) that is pointing to the interface generic-counters instead of the interface data-rate and that SELECT field (2) specifies a 1 second derivative, to be applied to the packet received field values.

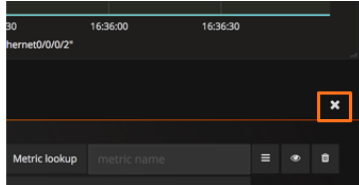


27. Click on the + button on the right of `derivative(1s)` to visualize the type native InfluxDB function. Select Transformations from the proposed menu, to visualize where `derivative` may be selected from.



NOTE: You can explore the list of InfluxDB native functions by accessing https://docs.influxdata.com/influxdb/v1.4/query_language/functions/.

28. Click on the X sign to exit the graph detail view.



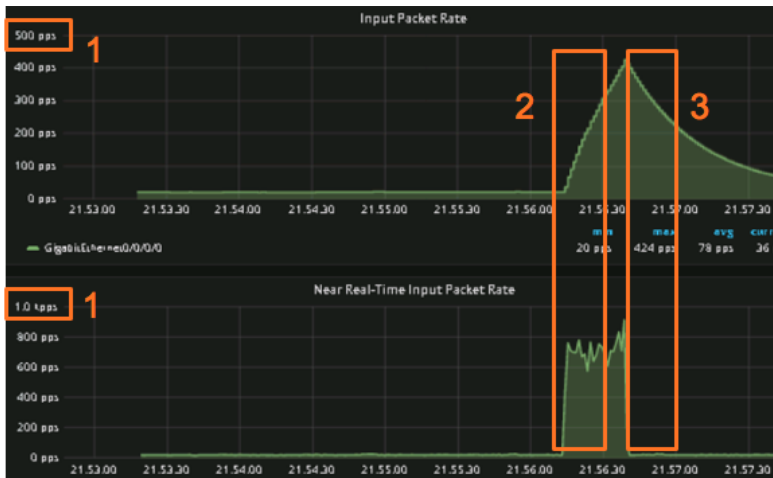
29. Double click on "Telemetry VM" icon to log on the Ubuntu server that collects the telemetry data.



30. Type `sudo ping 192.168.10.2 -s 10000 -f` in the SSH CLI to initiate a flood ping from the Linux server to the XRv router, to test the measurement sensitivity. Leave the ping running for few seconds and stop it with a CTRL-C command.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo ping 192.168.10.2 -s 10000 -f
PING 192.168.10.2 (192.168.10.2) 10000(10028) bytes of data.
. ^
--- 192.168.10.2 ping statistics ---
2376 packets transmitted, 2376 received, 0% packet loss, time 5209ms
```

31. Back to the Grafana dashboard, we can see a different picture in terms of perceived traffic: almost double the packet rate in this example for the near real-time (1) and a very different picture for the traffic profile. The input packet rate shows available resources (2) when they are actually utilized and used resources (3) when actually the interface is idled. Expect your system to report much more relaxed packet and data rates because they collect every 30s or minute at best (versus pushing the sliding averages pushed every 1 second like in this example).



When automating and self-healing the infrastructure, a near-real metric will be soon critical. For example, an optimization rerouting algorithm may need to assure that that a newly proposed path has actual spare capacity and it is not suffering of micro congestion, hidden by relaxed average traffic metrics.

Section 9: Exploring near real time measurements using Ostinato

This section explains how to generate traffic using the Ostinato application. This can be useful, if you decide to create more involving demonstrations with customized flows but if you are running short of time in this lab, you should consider to skip this section and move to the streaming of [routing metrics](#).

Ostinato is an open source application, that you can investigate at ostinato.org website and it provides an effective solution to generate traffic profiles, when functionalities of commercial solutions are not required. You can install Ostinato directly from your distribution (Linux option), you can compile from source or use the binaries from <http://ostinato.org/downloads> (available for few dollars donation).

1. Issue the command `sudo drone` from the Telemetry VM CLI (used for the extended ping), to activate Ostinato backend (Drone process).

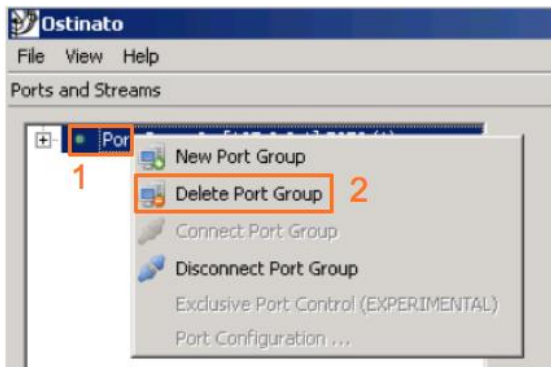
```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo drone
Retrieving the device list from the local machine

0. eth0
cap file = /tmp/qt temp.J17214
adding dev to all ports list <eth0>
< . . . SNIP >
The server is running on 0.0.0.0: 7878
Version: 0.8
Revision: a7bfec8ecb93@
Updater: state 2
Updater: state 3
Updater: state 4
Updater: HTTP/1.1 200 OK
Updater: state 5
Updater: latest version = 0.9
New Ostinato version 0.9 available. Visit http://ostinato.org to download
```

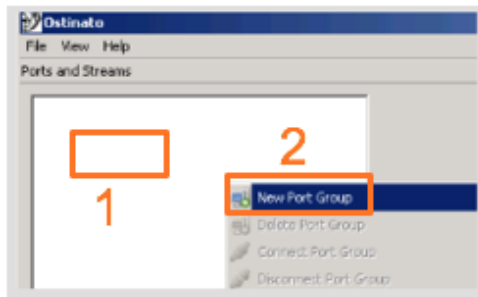
2. Double click on the Ostinato Icon to launch the application frontend. This is an open source application that can be useful for creative demonstration.



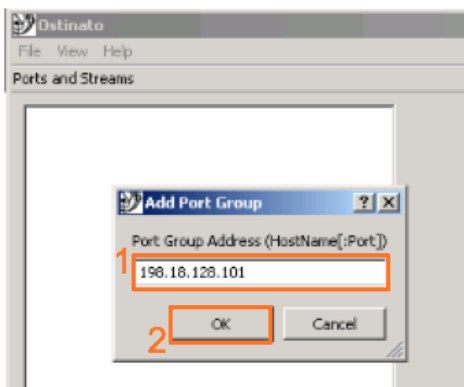
- By default, Ostinato launches the frontend and an instance of drone backend on the local machine. Right-click on the proposed default `port-group 0` (1), that point to the local drone process on Window and select `Delete Port Group` (2), from the proposed menu.



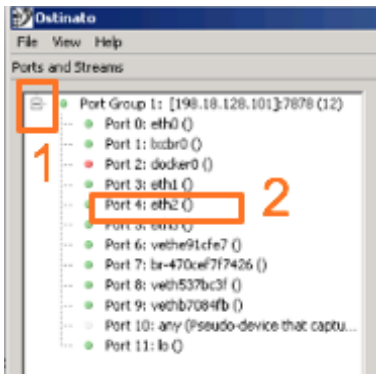
- Right-click on an empty section of `Ports and Streams` section (1) and select `New Port Group` (2), from the proposed menu.



- Type `198.18.128.101` in the `Add Port Group` text field and click on `OK`, to create a connection to the remote `drone` process running on the Telemetry VM.

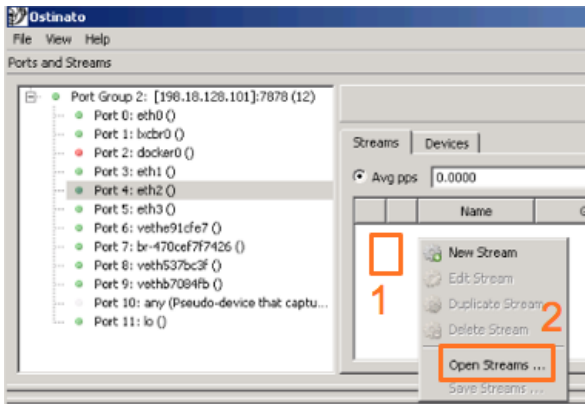


6. Click on the + sign (1) on the left of the Port Group 1 name, to expand the list of the remote interfaces and click on eth2 () (2).

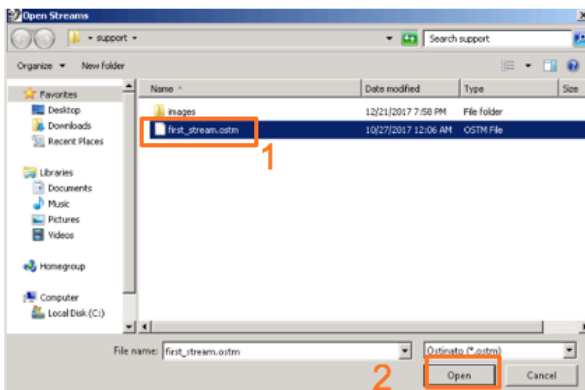


NOTE: The Telemetry VM is directly connected to the XRv router using two virtual patches. More specifically, eth2 is connected to GigabitEthernet 0/0/0/1 and eth3 is connected to GigabitEthernet 0/0/0/2.

7. Right-click on eth2 () flow details empty space (1) and select Open Streams ... (2) to select a pre-configured stream to load for this environment.

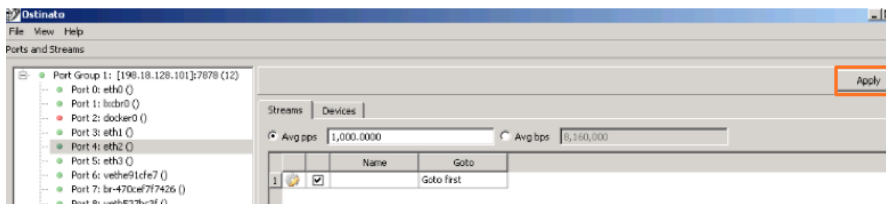


8. Click on first_stream.ostm (1) to select the file and then, click on the Open button to confirm the action. If the window doesn't show the first_stream.ostm file in the proposed default view, navigate to the Desktop/support directory.



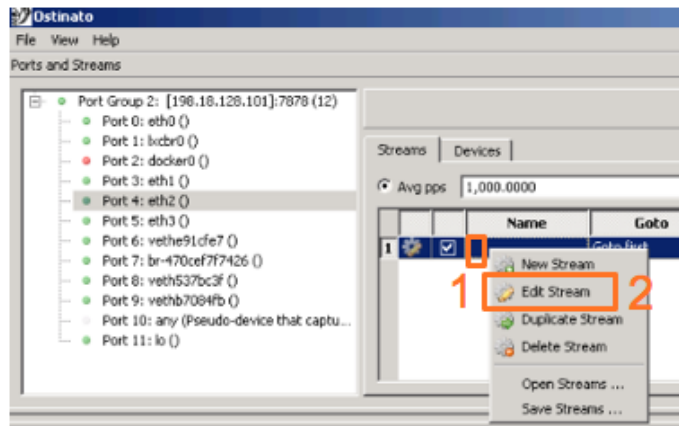
NOTE: `first stream.ostm` is a pre-configured data stream that sends 1k pps, 1k Bytes Ethernet frames from 1.1.1.2 (IP address assigned to the Linux server Eth2 interface) to 2.2.2.2 (IP address assigned to the Linux server Eth3 interface), that the XRv router will route (more information on how to explore these configuration in few steps).

9. **IMPORTANT** - Click on the `Apply` button to apply this traffic stream to the interface.

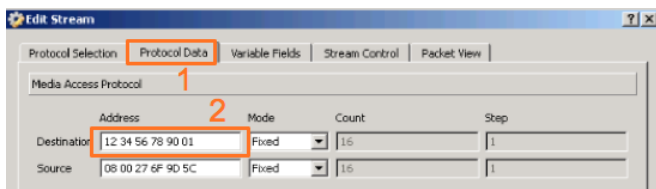


IMPORTANT NOTE: be careful not to skip this step when creating, importing or modifying any stream or nothing will be configured on the remote drone application. If you are not sure having clicked – click again.

10. Right-click on the newly imported flow (1) and select `Edit Stream` to appreciate the tool options and change for example rates or packet size.

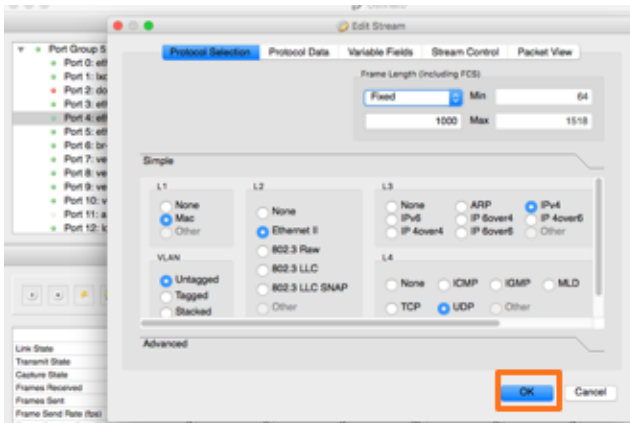


11. Click on Protocol Data (1) to check the stream L2 configuration and observe that a static MAC address (2) has been configured for the next hop router.

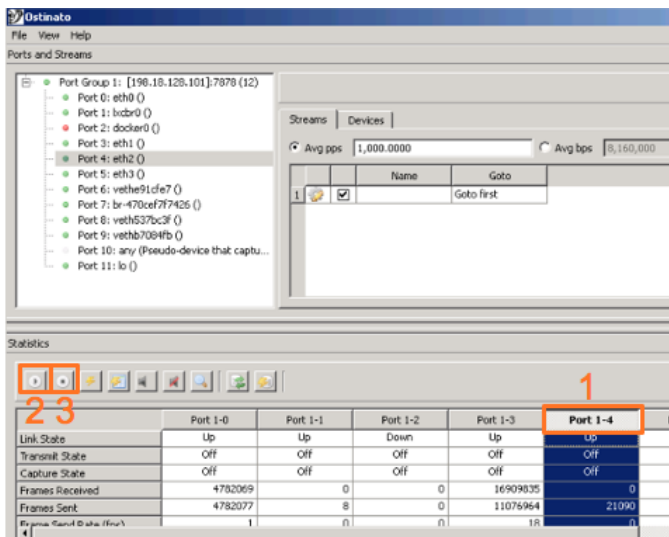


NOTE: Drone doesn't implement ARP resolution and if you want the router to process the traffic like in this example, you must statically define the MAC addresses in the flow details.

12. Close the stream detail window by clicking on OK (if you want to confirm any change) or just Cancel.

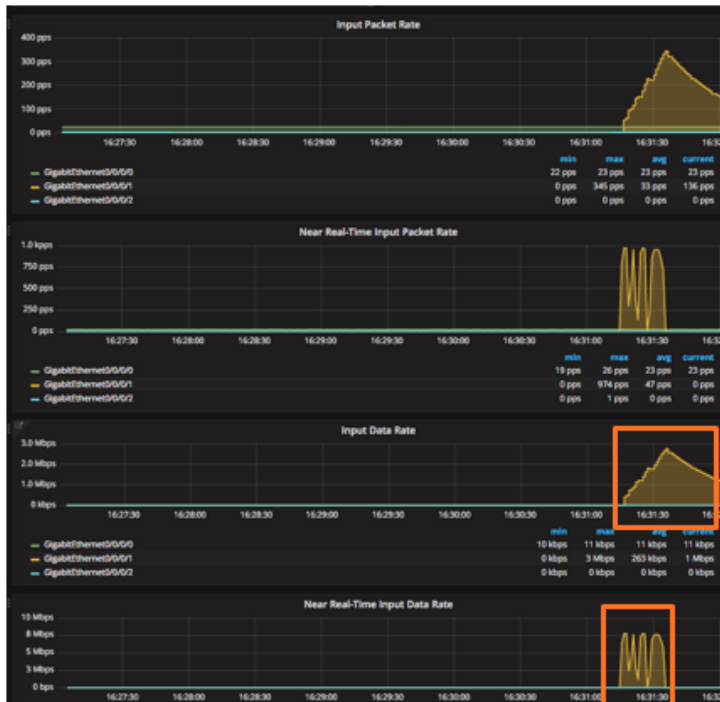


13. Click on Port 1-X, where X is the port allocated for eth2 or port 4 in our example (1) to select the just configured interface and start generating the stream by clicking on Start button (2), wait few second click on the stop button (3) and repeat this starting/stopping the flow few times.



NOTE: Notice that the Ostinato traffic tables reports detailed in information about the traffic send, in our example on Port1-4 (by our stream) and this traffic is received on Port1-5, after been routed by the XRv.

14. Check again the environment reference dashboard (explored in the previous section). As already discussed, near real-time graphs provide clear insight on the actual traffic and resource availability, versus the best average rates currently retrieved from the network.



NOTE: if you don't see traffic reported for `GigabitEthernet 0/0/0/1` and `GigabitEthernet 0/0/0/2`, ensure that you have selected them in the interface template field (top left of corner of the dashboard) per our earlier steps.

15. Stop the Drone process by issuing a CTRL-C command in the Telemetry VM SSH and close this session, by typing `exit`.

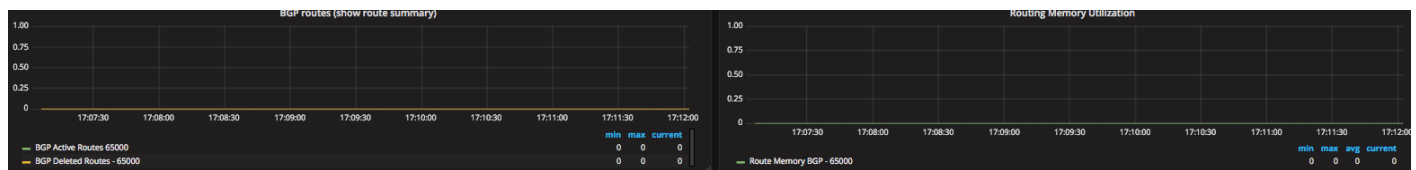
```
...
Receive stop requested but is not running!
In ~LinuxPort
In ~PcapPort
Receive stop requested but is not running!
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
```

Section 10: Streaming Routing Metrics

This section deals with a common requirement to retrieve information about the routing processes and more specifically, about the BGP route summary and information about memory utilization for store these routes.

We will use an `Exabgp` to peer with our XRV test router and to generate the BGP routes. `Exabgp` is a handy open-source Python project, that you can check for more information at <https://github.com/Exa-Networks/exabgp>.

1. Check the lower section of the environment reference dashboard (imported in [section 8](#)), for the BGP route summary (reporting on the information available in `show route summary`) and the routing memory utilization graphs. Explore these two graphs, using the information learned in [section 7](#) and [section 8](#) (click on the graph title, select Edit ...).



NOTE: Overlapping these two measurements provides a good indication of BGP routing memory leaks.

2. Double click on “Telemetry VM” icon to log on the Ubuntu server that collects the telemetry data.

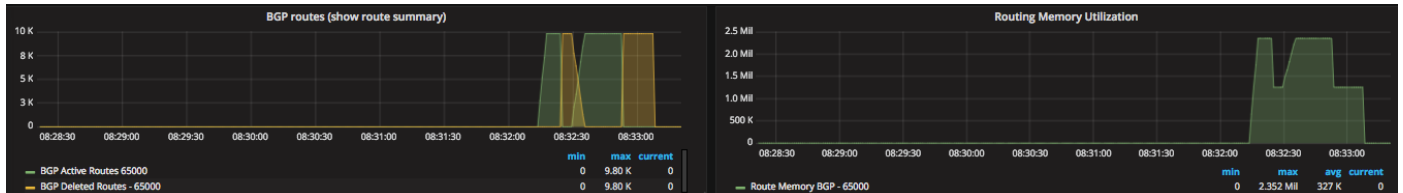


3. Issue the command `/vagrant/inject-bgp.sh` to create the BGP peer and start injecting routes toward the XRV router. Wait for the for the “Updated peers dynamic routes successfully” to be displayed, CTRL-C the program and launch `/vagrant/inject-bgp.sh` again, after few seconds. This procedure emulates a BGP peering bounce.

```
vagrant@vagrant-ubuntu-trusty-64:~$ /vagrant/inject-bgp.sh
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | environment file missing
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | generate it using "exabgp --fi >
/usr/local/etc/exabgp/exabgp.env"
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | reactor       | Performing reload of exabgp 3.4.17
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading | neighbor 192.168.10.2 {
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading | router-id 1.2.3.4;
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading | local-address
192.168.10.3;
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading | local-as 65000;
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading | peer-as 65000;
. . .
<SNIP for ~10k routes advertised>
. . .
Mon, 16 Jan 2017 06:31:45 | INFO      | 8378 | reactor       | Route added to neighbor 192.168.10.2
local-ip 192.168.10.3 local-as 65000 peer-as 65000 router-id 1.2.3.4 family-allowed in-open :
19.99.0.0/24 next-hop 192.168.10.3
Mon, 16 Jan 2017 06:31:46 | INFO      | 8378 | reactor       | Performing dynamic route update
Mon, 16 Jan 2017 06:31:46 | INFO      | 8378 | reactor       | Updated peers dynamic routes
successfully
Mon, 16 Jan 2017 06:31:51 | INFO      | 8378 | reactor       | ^C received
Mon, 16 Jan 2017 06:31:51 | INFO      | 8378 | reactor       | Performing shutdown
Mon, 16 Jan 2017 06:31:51 | INFO      | 8378 | processes     | Terminating process announce-routes
```

```
vagrant@vagrant-ubuntu-trusty-64:~$
vagrant@vagrant-ubuntu-trusty-64:~$ /vagrant/inject-bgp.sh
< . . . SNIP>
```

- Check the BGP summary route graph. It reports the peering bouncing, by changing the status of the initial almost 10k active BGP routes from active, to deleted state and to active again, when the peering is re-established. Notice also that the memory curve, following the same profile and fully releasing the allocated memory, when you stop the BGP peering by issuing a CTRL-C to the `inject-bgp.sh` python script.



- Cisco-IOS-XR-ip-rib-ipv4-oper YANG model that specifies the BGP route counts structure is not included in the default Pipeline's `metrics.json` file (describing which metrics, Pipeline needs to export and how to format the data). Use the `grep` command in the example, to visualize how `~/environment/metrics.json` has been updated to processes these metrics.

[Section 12 - Understaing Metric.json](#) provides more information about working with `metrics.json`.

```
vagrant@vagrant-ubuntu-trusty-64:~$ grep Cisco-IOS-XR-ip-rib-ipv4-oper -m 1 -A 16 -B 1
~/environment/metrics.json
{
  "basepath" : "Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-
names/ip-rib-route-table-name/protocol/ospf/as/information",
  "spec" : {
    "fields" : [
      {"name" : "vrf-name", "tag" : true},
      {"name" : "af-name", "tag" : true},
      {"name" : "saf-name", "tag" : true},
      {"name" : "route-table-name", "tag" : true},
      {"name" : "as", "tag" : true},
      {"name" : "routes-counts"},
      {"name" : "active-routes-count"},
      {"name" : "deleted-routes-count"},
      {"name" : "paths-count"},
      {"name" : "protocol-route-memory"},
      {"name" : "backup-routes-count"}
    ]
  }
},
```

Optional steps – if you are interested to personalize the BGP peer, you can update with a command editor of choice, `conf.ini` for the peering configuration options and/or the Python script specified in the `process announce-route`, that as explained in the example below, implements a loop to advertise few thousands BGP prefixes to the router.

- Issue the command `cat /vagrant/inject-bgp.sh` to visualize how exabgp is initiated by passing the configuration file `/vagrant/bgp/conf.ini`.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat /vagrant/inject-bgp.sh
exabgp /vagrant/bgp/conf.ini
```

7. Issue `cat /vagrant/bgp/conf.ini` to visualize the BGP peering information

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat /vagrant/bgp/conf.ini
neighbor 192.168.10.2 {
    router-id 1.2.3.4;
    local-address 192.168.10.3;
    local-as 65000;
    peer-as 65000;
    #graceful-restart;

    process announce-routes {
        run /vagrant/bgp/api-add-remove.run;
    }
}
```

8. Issue the command `cat /vagrant/bgp/api-add-remove.run`, to visualize how a basic loop in a Python script, writes the routes to advertise to stdout (standard output), that exabgp interprets as “to advertise”.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat /vagrant/bgp/api-add-remove.run
#!/usr/bin/env python

import sys
import time
next_hop="192.168.10.3"

for x in range (1, 100):
    for y in range (1, 100):
        sys.stdout.write("announce route "+str(x)+"."+str(y)+".0.0/24 next-hop "+next_hop +'\n')
        sys.stdout.flush()
while True:
    time.sleep(1)
```

NOTE: The highlighted double loop, advertises the following routes:

```
announce route 1.1.0.0/24 next-hop 192.168.10.3
announce route 1.2.0.0/24 next-hop 192.168.10.3
<>
announce route 99.99.0.0/24 next-hop 192.168.10.3
```

9. Close all the active SSH sessions and the Chrome browser.

Section 11: Exploring Kapacitor

In this section, we are going to configure a KPI policy in Kapacitor, that will trigger alerts based on the CPU utilization.

“Kapacitor is a native data processing engine that can process both stream and batch data from InfluxDB. It lets you plug in your own custom logic or user-defined functions to process alerts with dynamic thresholds, match metrics for patterns, compute statistical anomalies, and perform specific actions based on these alerts like dynamic load rebalancing.”

- <https://www.influxdata.com/time-series-platform/kapacitor/>

Kapacitor uses TICK scripts to define data processing pipelines. A pipeline is set of nodes that process data and edges that connect the nodes. Pipelines in Kapacitor are directed acyclic graphs (DAGs) meaning each edge has a direction that data flows and there cannot be any cycles in the pipeline.

Each edge has a type, one of the following:

- StreamEdge – an edge that transfers data a single data point at a time.
- BatchEdge – an edge that transfers data in chunks instead of one point at a time.
- <https://docs.influxdata.com/kapacitor/v1.4/tick/>

In the example proposed, we are going to create a basic processing pipeline (sorry for term overloading), that consumes a stream of data from InfluxDB and we will further explore the DAG concept.

This environment launches an instance of the Kapacitor, when we started InfluxDB and Grafana in [section 4 – Staging a Telemetry Stack](#), using a common docker-compose initialization file.

```
kapacitor:
  restart: always
  image: kapacitor:1.3
  environment:
    KAPACITOR_HOSTNAME: kapacitor
    KAPACITOR_INFLUXDB_0_URLS_0: http://influxdb:8086
  volumes:
    - ~/data/kapacitor:/var/lib/kapacitor
    - ~/log/kapacitor:/tmp
  links:
    - influxdb
  ports:
    - "9092:9092"
```

Notes about Kapacitor container configuration:

- This lab has been tested using Kapacitor version 1.3 but you are free to change to a newer version by updating the image numeric tag (you should align InfluxDB and Kapacitor versions).
- Using `KAPACITOR_INFLUXDB_0_URLS_0` variable, we are connecting Kapacitor to the running instance of InfluxDB. This configuration is important because it generates in InfluxDB a default subscription, that sends a copy of the data received by InfluxDB to Kapacitor.
- Kapacitor logs have been mounted on the host in the directory `~/log/kapacitor`. Later, in the section we are going to configure policies that will log in this directory threshold notifications.

The `docker-compose` file also describes a `kapacitor-cli` container, linked to the actual `kapacitor` container, that provides an interactive CLI (in the same way we have used InfluxDB-cli earlier in the document).

```

kapacitor-cli:
  image: kapacitor:1.3
  entrypoint: bash
  environment:
    KAPACITOR_URL: http://kapacitor:9092
  volumes:
    - ~/environment/ticks:/root/ticks
  links:
    - kapacitor

```

1. Double click on “Telemetry VM” icon to log on the Ubuntu server that collects the telemetry data.



2. Type `cd ~/environment && docker-compose run kapacitor-cli` to run an interactive Kapacitor’s CLI container.

```

vagrant@vagrant-ubuntu-trusty-64:~$ cd ~/environment && docker-compose run kapacitor-cli
WARNING: The http_proxy variable is not set. Defaulting to a blank string.
WARNING: The https proxy variable is not set. Defaulting to a blank string.
WARNING: The no proxy variable is not set. Defaulting to a blank string.
Starting environment influxdb 1 ... done
Starting environment kapacitor 1 ... done
root@3f32f784597a:/#

```

3. Type `kapacitor stats ingress` to inspect the data sent from InfluxBD to Kapacitor in terms of data points.

Notice in the last section of the output, that we have one entry for each of the five measurements under test (stored in `mdt_db` database with a retention of 6 hours).

```

root@3f32f784597a:/# kapacitor stats ingress

```

Database	Retention	Policy	Measurement	Points Received	
_internal	monitor	cq		800	
internal	monitor	database		1600	
internal	monitor	httpd		800	
internal	monitor	queryExecutor		800	
internal	monitor	runtime		800	
_internal	monitor	shard		3273	
_internal	monitor	subscriber		2400	
_internal	monitor	tsml_cache		3273	
internal	monitor	tsml_engine		3273	
_internal	monitor	tsml_filestore		3273	
_internal	monitor	tsml_wal		3273	
internal	monitor	write		800	
kapacitorautogen		edges		684	
kapacitorautogen		ingress		17976	
kapacitor autogen		kapacitor		805	
kapacitor autogen		nodes		1074	
kapacitor autogen		runtime		805	
kapacitor autogen		topics		358	
mdt_db	test6h	Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces		37625	
		/interface/latest/data-rate			
mdt_db	test6h	Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces			
		/interface/latest/generic/counters		37625	
mdt_db	test6h	Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf			

		/ip-rib-route-table-names/ip-rib-route-table-name	
		/protocol/bgp/as/information	7525
mdt db	test6h	Cisco-IOS-XR-nto-misc-oper:memory/summary/nodes/node/summary	7525
mdt_db	test6h	Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization	2001829

4. Type `kapacitor list tasks` to confirm that there are no pre-configured policies.

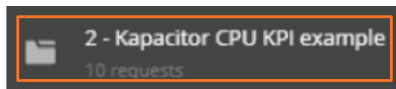
```
root@3f32f784597a:/# kapacitor list tasks
```

ID	Type	Status	Executing Databases and Retention Policies

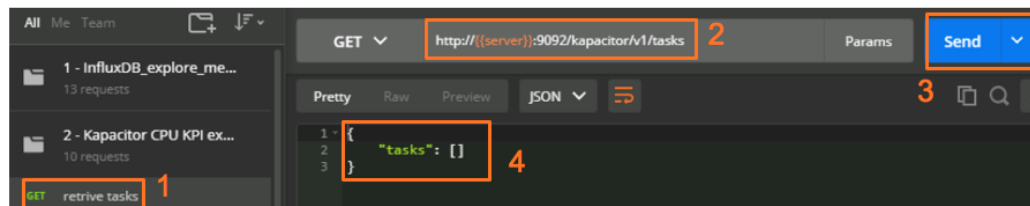
5. Double click on the Postman icon to launch the program. We are going to configure Kapacitor using REST APIs, to explain how this can be integrated in automation tools. Alternatively, Kapacitor can be configured directly from CLI as described in https://docs.influxdata.com/kapacitor/v1.4/introduction/getting_started.



6. Click on the `2 - Kapacitor CPU KPI example` collection to visualize the list of preconfigured REST calls.

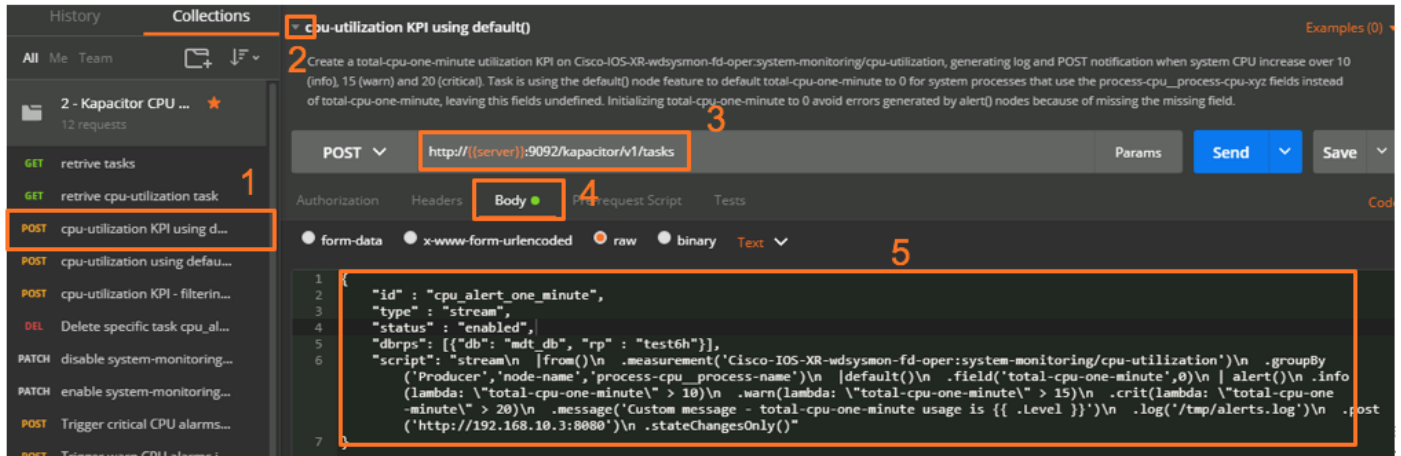


7. Click on `retrieve tasks` (1) to visualize the details of the GET API (2) `http://{{server}}:9092/kapacitor/v1/tasks`, click on the Send button (3) to trigger the REST call and finally and finally, review the query output (reported as Pretty JSON in 4).



NOTE: As expected, this REST GET query returns the same empty list as the Kapacitor CLI command that we have just explored in step 4. Kapacitor provides a 1:1 mapping between CLI command and REST calls, which is key for automation.

8. Click on `cpu-utilization KPI using default()` REST call (1) example to visualize the POST details. Click on the expansion sign (2), to read the POST description. Check the POST API (3) suggested to create a new task and finally, click on `body` (4) to visualize the KPI policy, that will be sent with this task (5).



NOTE: when creating a policy in Postman, you cannot break the script string in multiple lines. If you do it, you will receive an error message like "error": "invalid JSON". As in the example, the script must be a single line and you need to code the required new lines as "\n".

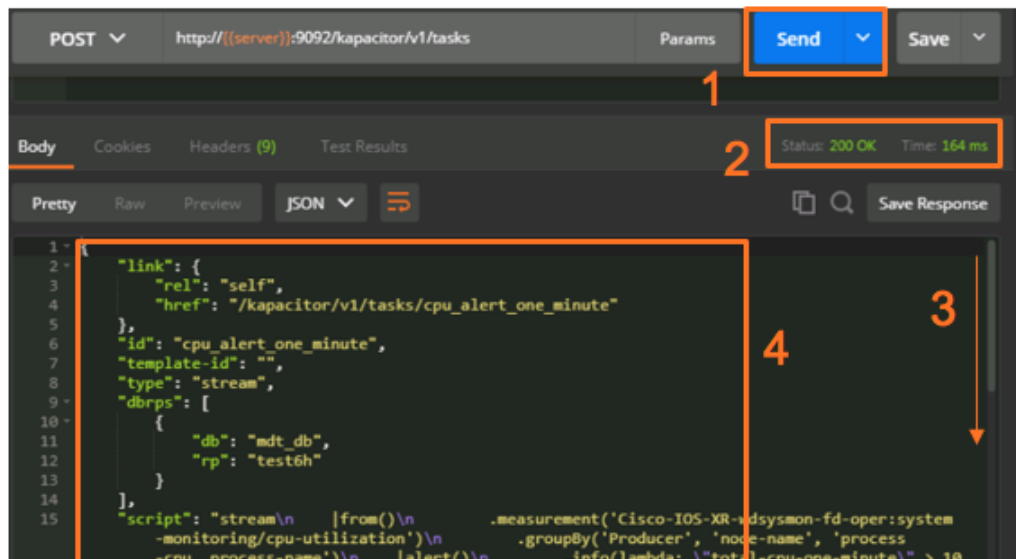
The following policy describes the same configuration formatted as pretty JSON. This actual file can be loaded using the Kapacitor CLI and it would create the same KPI policy.

```
{
  "id": "cpu alert one minute",
  "type": "stream",
  "status": "enabled",
  "dbrps": [{"db": "mdt db", "rp": "test6h"}],
  "script": "stream
    | from()
      .measurement('Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization')
      .groupBy('Producer','node-name','process-cpu__process-name')
    | default()\n
      .field('total-cpu-one-minute',0)
    | alert()
      .info(lambda: \"total-cpu-one-minute\" > 10)
      .warn(lambda: \"total-cpu-one-minute\" > 15)
      .crit(lambda: \"total-cpu-one-minute\" > 20)
      .message('Custom message - total-cpu-one-minute usage is {{ .Level }}')
      .log('/tmp/alerts.log')
      .post('http://192.168.10.3:8080')
      .stateChangesOnly() "
}
```

The DAG graph proposed for this policy consist of a basic 4 nodes pipeline:

- `Stream()` Node
 - o Initial edge node receiving the data from Influx.
- Followed by a `From()` Node
 - o Select measurement from the stream
- Followed by a `Default()` Node ()
 - o Initialize `total-cpu-one-minute` to 0 for Processes CPU measurements (reporting on `process-cpu__process-cpu-xyz` field instead of `total-cpu-one-minute`) to avoid the Alert node to log an error, when processing an empty field.
- Followed by a final, `Alert()` node
 - o Define 4 thresholds (ok, info, warm and critical) for the `total-cpu-one-minute` measurement field value
 - o Specify the threshold messages to send to the Kapacitor alert log file and to POST to an example web server (described later in the section), but only when the KPI status change.

9. Click on the `Send` button to trigger the POST REST call (1), validate the return code (2) that should be 200 for a successful POST calls, slide the window (3) to visualize the returned JSON description (4), with detailed information about the created policy (these details will be explained in the next steps, when query a policy).



10. Issue again the `retrieve tasks` GET REST call (explored in 7) to visualize the newly created policy and its details, including the created DAG graph and the number of data points, processed by each node (handy to check if a policy is actually working as expected). You may decide to check these counter in the next few steps because the CLI provide a better formatting.
11. Type `kapacitor list tasks` in the Kapacitor CLI to confirm that the newly create `cpu_alert_one_minute` policy is properly reported.

```
root@3f32f784597a:/# kapacitor list tasks
ID                Type      Status   Executing Databases and Retention Policies
cpu_alert_one_minute stream    enabled  true      ["mdt_db"."test6h"]
```

12. Type `kapacitor show cpu_alert_one_minute` in the Kapacitor CLI to visualize the create `cpu_alert_one_minute` policy details, including the counters we have discussed in the previous steps.

```

root@60329e9798e9:/# kapacitor show cpu_alert_one_minute
ID: cpu alert one minute
Error:
Template:
Type: stream
Status: enabled
Executing: true
Created: 28 Dec 17 23:14 UTC
Modified: 28 Dec 17 23:14 UTC
LastEnabled: 28 Dec 17 23:14 UTC
Databases Retention Policies: ["mdt db"."test6h"]
TICKscript:
stream
  |from()
    .measurement('Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization')
    .groupBy('Producer', 'node-name', 'process-cpu__process-name')
  |default()
    .field('total-cpu-one-minute', 0)
  |alert()
    .info(lambda: "total-cpu-one-minute" > 10)
    .warn(lambda: "total-cpu-one-minute" > 15)
    .crit(lambda: "total-cpu-one-minute" > 20)
    .message('Custom message - total-cpu-one-minute usage is {{ .Level }}')
    .log('/tmp/alerts.log')
    .post('http://192.168.10.3:8080')
    .stateChangesOnly()

DOT:
digraph cpu alert one minute {
graph [throughput="266.00 points/s"];

stream0 [avg_exec_time_ns="0s" errors="0" working_cardinality="0" ];
stream0 -> from1 [processed="299516"];

from1 [avg exec time ns="9.84µs" errors="0" working cardinality="0" ];
from1 -> default2 [processed="299516"];

default2 [avg exec time ns="16.585µs" errors="0" fields defaulted="298390" tags defaulted="0"
working cardinality="0" ];
default2 -> alert3 [processed="299516"];

alert3 [alerts triggered="2" avg exec time ns="45.501µs" crits triggered="0" errors="0"
infos_triggered="0" oks_triggered="0" warns_triggered="0" working_cardinality="249" ];
}

```

13. Type `kapacitor list topics` in the Kapacitor CLI, for a nice compact representation of all policies configures including their current threshold levels and collected level transition (explained later in the section).

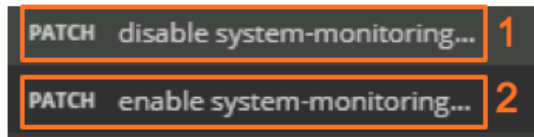
```

root@3f32f784597a:/# kapacitor list topics
ID                               Level    Collected
main:cpu_alert_one_minute:alert3 OK        0

```

NOTE: a different level may be reported, if the router CPU is running already high in the dCloud environment.

14. (Optional) Using the same Postman procedure discussing in the first part of this section (selecting a REST call example, investigating the REST string and clicking on the SEND button), you can explore two useful PATCH REST calls that disable (disable system-monitoring/cpu-utilization) and re-enable (enable system-monitoring/cpu-utilization) the previously configured policy, without deleting it from the system.



15. Double click on "Telemetry VM" icon to open a new session to the Telemetry Ubuntu server. We going to use to use this new session to visualize the Kapacitor KPI notifications.



16. Type the command `~/environment/web-alarm.py` in the newly create SSH session, to launch a basic Python web server that you can easily personalize for your test use cases.

Move this window on the side of screen, where you can still check for incoming notifications.

```
vagrant@vagrant-ubuntu-trusty-64:~$ environment/web-alarm.py
http://0.0.0.0:8080/
```

The web-alarm.py application uses the web Python module to implement the WEB server and by default binds to the local IP addresses, at the TCP address 8080. Kapacitor will sent KPI notifications as POST REST and this application will process them in the POST() function, by printing a different message, based on the alarm type.

You may consider to change the POST() function to personalize your use cases.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat ~/environment/web-alarm.py
#!/usr/bin/python2
import web

urls = ('/', 'index')

class index:
    def GET(self):
        return "Hello, world!"

    def POST(self):
        data = web.data()
        print
        print 'HTTP POST RECEIVED:'
        print data
        print
        if 'CRITICAL' in data:
            print "Alert Received: Critical Alarm received"
        elif 'WARN' in data:
            print "Alert Received: Warm Alarm received"
        elif 'INFO' in data:
            print "Alert Received: Info Alarm received"
```

```

elif 'OK' in data:
    print "Alert Received: Alarm cleared"
else:
    print "Alert Received but I don't understand the data"

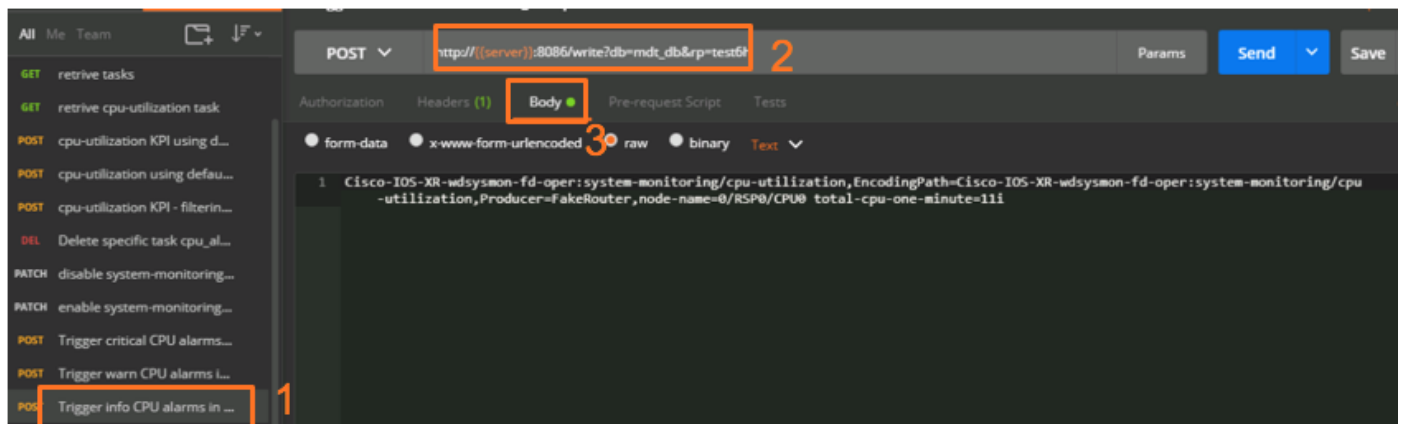
    return #'OK'

if name == " main ":
    app = web.application(urls, globals())
    app.run()

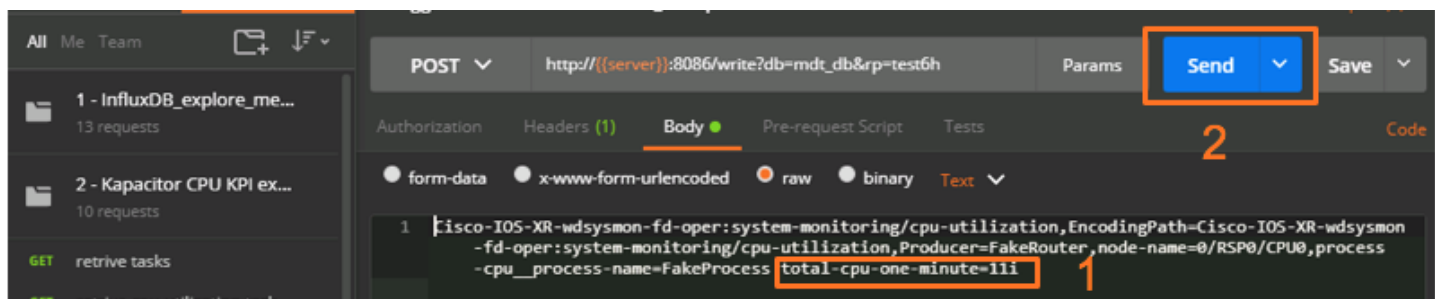
```

NOTE: The next steps validate the Kapacitor policy just created, by increasing the CPU utilization. We can achieve this outcome using a flood ping from the telemetry server toward the router (i.e. `sudo ping 192.168.10.2 -f -s 10000`) or using Ostinato, as explained in the previous sections. The next steps suggest a different strategy to test the policy, inject synthetic data to validate each threshold in the KPI policy. This procedure may be extremely useful to automate your testing process.

- Click on `Trigger info CPU alarms` call example (1) (or higher utilization, for example `Trigger warm CPU alarm`, if your environment already notified an info state in step 13). Check the POST call definition (2) and finally, click on body (3) to visualize the data point details. Notice in (2) that we will write on the `mdt_db`, requesting a retention policy of 6 hours (previously discussed, when exploring the InfluxDB).



- Check the body details (1), you should now be familiar with the measurement, tags and fields values required for a single data point in CPU utilization. See that the `total-cpu-one-minute` is specified equal to 11 (or 11i as integer) which is higher than 10 specified for the info threshold alarm. Finally click the Send button (2) to send this REST call toward InfluxDB.



- Check the SSH session window, where you launched the `web-alarm.py` utility, few steps before. You should see an info alarm, created by Kapacitor with the information specified in the just created synthetic data point.

```
vagrant@vagrant-ubuntu-trusty-64:~$ environment/web-alarm.py
```

```
http://0.0.0.0:8080/
```

```
HTTP POST RECEIVED:
```

```
{ "id": "Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization:Producer=FakeRouter,node-
name=0/RSP0/CPU0,process-cpu__process-name=FakeProcess", "message": "Custom message - total-cpu-one-minute
usage is INFO", "details": "{\u0026#34;Name\u0026#34;:\u0026#34;Cisco-IOS-XR-wdsysmon-fd-oper:system-
monitoring/cpu-
utilization\u0026#34;,\u0026#34;TaskName\u0026#34;:\u0026#34;cpu_alert_one_minute\u0026#34;,\u0026#34;Gr
oup\u0026#34;:\u0026#34;Producer=FakeRouter,node-name=0/RSP0/CPU0,process-cpu__process-
name=FakeProcess\u0026#34;,\u0026#34;Tags\u0026#34;:{\u0026#34;EncodingPath\u0026#34;:\u0026#34;Cisco-
IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-
utilization\u0026#34;,\u0026#34;Producer\u0026#34;:\u0026#34;FakeRouter\u0026#34;,\u0026#34;node-
name\u0026#34;:\u0026#34;0/RSP0/CPU0\u0026#34;,\u0026#34;process-cpu__process-
name\u0026#34;:\u0026#34;FakeProcess\u0026#34;},{\u0026#34;ServerInfo\u0026#34;:{\u0026#34;Hostname\u0026
#34;:\u0026#34;kapacitor\u0026#34;,\u0026#34;ClusterID\u0026#34;:\u0026#34;c891ded1-8a77-4ca9-8fd6-
a5ac70814b8b\u0026#34;,\u0026#34;ServerID\u0026#34;:\u0026#34;f91ca338-a355-4eec-bf3e-
d8ef621eb42d\u0026#34;},{\u0026#34;ID\u0026#34;:\u0026#34;Cisco-IOS-XR-wdsysmon-fd-oper:system-
monitoring/cpu-utilization:Producer=FakeRouter,node-name=0/RSP0/CPU0,process-cpu__process-
name=FakeProcess\u0026#34;,\u0026#34;Fields\u0026#34;:{\u0026#34;total-cpu-one-
minute\u0026#34;:11},{\u0026#34;Level\u0026#34;:\u0026#34;INFO\u0026#34;,\u0026#34;Time\u0026#34;:\u0026#
34;2017-12-19T02:48:09.787454281Z\u0026#34;,\u0026#34;Message\u0026#34;:\u0026#34;Custom message -
total-cpu-one-minute usage is INFO\u0026#34;}\n", "time": "2017-12-
19T02:48:09.787454281Z", "duration": 0, "level": "INFO", "data": {"series": [{"name": "Cisco-IOS-XR-wdsysmon-fd-
oper:system-monitoring/cpu-utilization", "tags": {"EncodingPath": "Cisco-IOS-XR-wdsysmon-fd-oper:system-
monitoring/cpu-utilization", "Producer": "FakeRouter", "node-name": "0/RSP0/CPU0", "process-cpu__process-
name": "FakeProcess"}, "columns": ["time", "total-cpu-one-minute"], "values": [{"2017-12-
19T02:48:09.787454281Z", 11}]}]}}
```

```
Alert Received: Info Alarm received
```

```
172.18.0.4:54579 - - [19/Dec/2017 02:48:09] "HTTP/1.1 POST /" - 200 OK
```

NOTE: You can send this measurement multiple times and you will notice that no more notifications are generated because we have configured our policy to send a notification when there is a change of state.

20. Issue the command `kapacitor list topic` in the Kapacitor CLI window, to see how the topic summary report a INFO level (or higher based on the data point injected) and has collect the alarm generated.

```
root@3f32f784597a:/# kapacitor list topics
ID                               Level    Collected
main:cpu_alert_one_minute:alert2 INFO      1
```

21. Issue the command `kapacitor show-topic main:cpu_alert_one_minute:alert<ID>` (using the alert ID retrieved in the previous step) for more information about the event.

```
root@78712ac38835:/# kapacitor show-topic main:cpu_alert_one_minute:alert2
ID: main:cpu alert one minute:alert2
Level: INFO
Collected: 1
Handlers: []
Events:
Event      Level    Message                                                                                               Date
Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization:Producer=FakeRouter,node-
name=0/RSP0/CPU0,process-cpu__process-name=FakeProcess
INFO      Custom message - total-cpu-one-minute usage is INFO      19 Dec 17 02:48 UTC
```

22. Type `exit` in the Kapacitor CLI to stop the container

```
root@684f793f5000:/# exit
exit
vagrant@vagrant-ubuntu-trusty-64:~/environment$
```

23. Type `sudo tail ~/log/kapacitor/alerts.log` in the SSH session to the Telemetry VM, to validate (as requested in the KPI policy) that the same notification was also saved locally in the Kapacitor `alert.log` file.

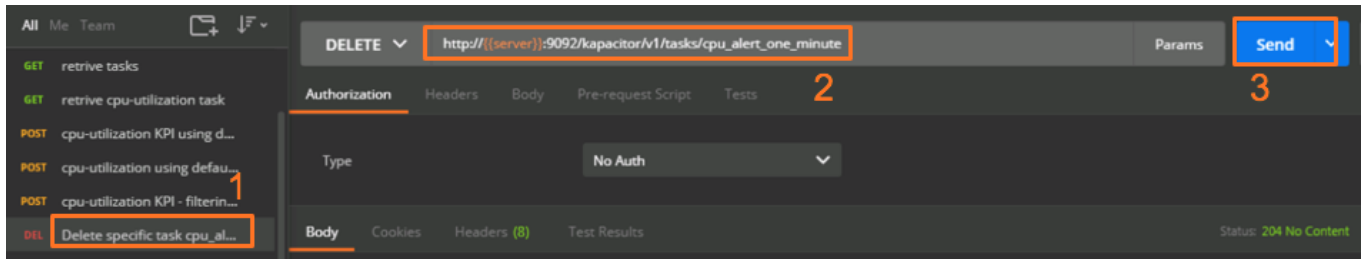
```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo tail ~/log/kapacitor/alerts.log
{"id":"Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization:Producer=FakeRouter,node-
name=0/RSP0/CPU0,process-cpu__process-name=FakeProcess","message":"Custom message - total-cpu-one-minute
usage is INFO","details":{"\u0026#34;Name\u0026#34;:\u0026#34;Cisco-IOS-XR-wdsysmon-fd-oper:system-
monitoring/cpu-utilization\u0026#34;
<SNIP>
"time":"2017-12-19T02:48:09.787454281Z","duration":0,"level":"INFO","data":{"series":[{"name":"Cisco-
IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization","tags":{"EncodingPath":"Cisco-IOS-XR-
wdsysmon-fd-oper:system-monitoring/cpu-utilization","Producer":"FakeRouter","node-
name":"0/RSP0/CPU0","process-cpu__process-name":"FakeProcess"},"columns":["time","total-cpu-one-
minute"],"values":[{"2017-12-19T02:48:09.787454281Z",11}]}]}}
```

24. (Optional) Using the same procedure used to generate an info CPU alarm, you can explore generating a warm alarm (1), a critical alarm (2) and finally clearing the alarm (3). Alternatively move to the next step.

```
POST Trigger critical CPU alarms in mdt_db&rp=test6h 2
POST Trigger warn CPU alarms in mdt_db&rp=test6h 1
POST Trigger info CPU alarms in mdt_db&rp=test6h
POST Trigger clear CPU alarms in mdt_db&rp=test6h 3
```

25. Click on the Delete specific task `cpu_alert_one_minute` (1), check the DELETE REST call provided to delete the previously configured Kapacitor policy and finally, click on the `Send` button (3) to delete the policy.

You can use any of the previously discussed procedure (by CLI or API) to validate that the policy was effectively removed from Kapacitor.



26. Close the Postman window and the rest of the SSH session windows before moving to the next section.

NOTE: What next with Kapacitor? [Chronograf](#) is another open source project (which is currently not covered in lab), that you may consider to explore, if you are interested in a UI framework to manage the live cycle of alarms in Kapacitor.

<https://docs.influxdata.com/chronograf/v1.3/guides/create-a-kapacitor-alert/>.

Also, consider the tutorial videos proposed in <https://www.influxdata.com/university>.

Section 12: Understanding metrics.json

Nicely described by Shelly Cadora in <https://xrdocs.github.io/telemetry/tutorials/2017-04-10-using-pipeline-integrating-with-influxdb>:

“One of the important functions of Pipeline is to take the hierarchical YANG-based data and transform it into the Line Protocol for easy consumption by Influxdb. Pipeline takes the complex, hierarchical YANG-modeled data and flattens it into multiple time series. Pipeline uses the metrics.json file to perform the transformation. The metrics.json file contains a series of json objects, one for each YANG model and sub-tree path that the router streams.”

At time of writing, there is no open source Cisco automated tool to generate the data structure for `metrics.json` from the YANG file.

Metrics that are not part of the standard Pipeline `metrics.json`, must be created manually for example from the unprocessed data collected in the Pipeline `dump.txt` file.

In this section, we are going to explain how we derived the `metric.json` object, for the already discussed `Cisco-IOS-XR-ip-rib-ipv4-oper` YANG model.

1. Double click on “Telemetry VM” icon to open a new session to the Telemetry Ubuntu server.



2. Issue the command `grep Cisco-IOS-XR-ip-rib-ipv4-oper -m 2 -A 35 -B 2 ~/log/dump.txt` to extract an example for the data, streamed by the IP RIB IPV4 sensor path. We will need the `encoding_path`, `Keys` and `Content` fields in few steps.

```
vagrant@vagrant-ubuntu-trusty-64:~$ grep Cisco-IOS-XR-ip-rib-ipv4-oper -m 2 -A 35 -B 2 ~/log/dump.txt
```

```
----- 2017-01-15 23:33:13.815305815 +0000 UTC -----
```

```
Summary: GPB(common) Message [192.168.10.2:55570(test_XR)/Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-name/protocol/bgp/as/information msg len: 559]
```

```
{
  "Source": "192.168.10.2:55570",
  "Telemetry": {
    "node_id_str": "test_XR",
    "subscription id str": "1",
    "encoding path": "Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information",
    "collection id": 5,
    "collection_start_time": 1484523194443,
    "msg timestamp": 1484523194443,
    "collection end time": 1484523194471
  },
  "Rows": [
    {
      "Timestamp": 1484523194461,
      "Keys": {
        "af-name": "IPv4",
        "as": "65000",
        "route-table-name": "default",
        "saf-name": "Unicast",
        "vrf-name": "default"
```

```

    },
    "Content": {
      "active-routes-count": 0,
      "backup-routes-count": 0,
      "deleted-routes-count": 0,
      "instance": "65000",
      "paths-count": 0,
      "protocol-clients-count": 1,
      "protocol-names": "bgp",
      "protocol-route-memory": 0,
      "redistribution-client-count": 0,
      "routes-counts": 0,
      "version": 0
    }
  }
}
]
}
----- 2017-01-15 23:33:14.735864978 +0000 UTC -----
Summary: GPB(common) Message [192.168.10.2:55570(test_XR)/Cisco-IOS-XR-wdsysmon-fd-oper:system-
monitoring/cpu-utilization msg len: 40841]

```

NOTE: The grep command support searching for a given string in a file and takes extra options like:

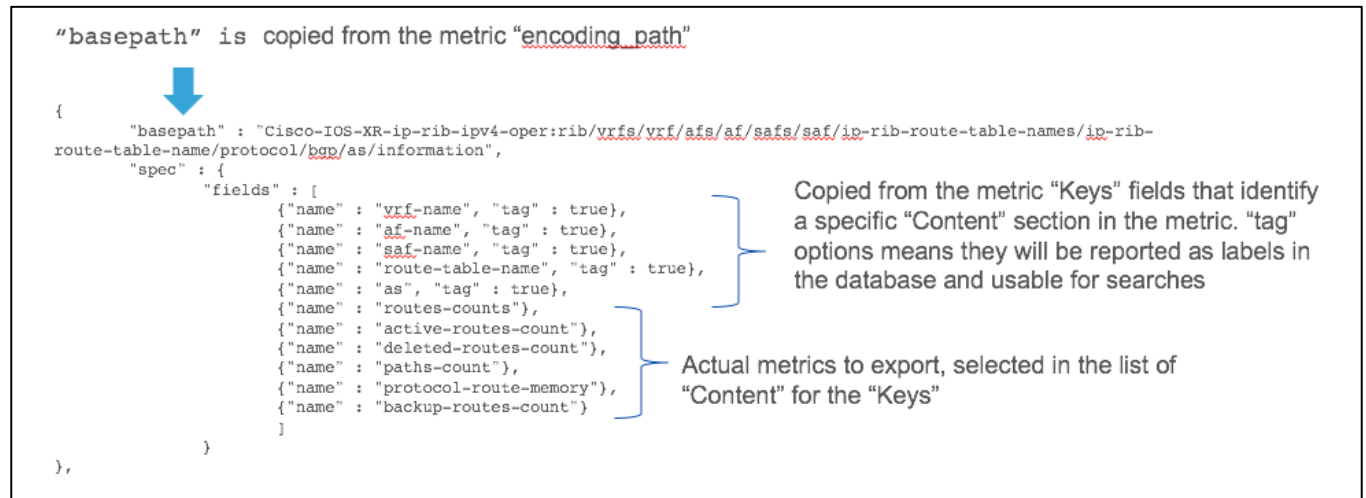
“A” to visualize A lines after the pattern – you will have to adjust if long metric

“B” to visualize B lines before the pattern – to include the pattern in the extract

“m” for the number of instances to search – in our case 2 because the pattern is repeated in the metric

- Copy one of the examples provided in `metrics.json` and substitute its field values with the values retrieved from the captured metric in the previous step. Please notice that some metrics are hierarchical and they require a structure similar to Cisco-IOS-XR-qos-ma-oper from the `metrics.json` examples.

“basepath” is copied from the metric “encoding_path”



```

{
  "basepath": "Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-rib-
route-table-name/protocol/bgp/as/information",
  "spec": {
    "fields": [
      { "name": "vrf-name", "tag": true },
      { "name": "af-name", "tag": true },
      { "name": "saf-name", "tag": true },
      { "name": "route-table-name", "tag": true },
      { "name": "as", "tag": true },
      { "name": "routes-counts", "tag": true },
      { "name": "active-routes-count", "tag": true },
      { "name": "deleted-routes-count", "tag": true },
      { "name": "paths-count", "tag": true },
      { "name": "protocol-route-memory", "tag": true },
      { "name": "backup-routes-count", "tag": true }
    ]
  }
}

```

Copied from the metric “Keys” fields that identify a specific “Content” section in the metric. “tag” options means they will be reported as labels in the database and usable for searches

Actual metrics to export, selected in the list of “Content” for the “Keys”

- Append this this new section in `metric.json`.
- Reload Pipeline for the changes in `metric.json` to be applied. You may use the `telemetry_utility.sh` and select option 4 (Re-start Pipeline).

```
vagrant@vagrant-ubuntu-trusty-64:~$ ./telemetry_utility.sh
```

```
1) Verify Environment
2) Destroy & Clean Environment
3) Start/Stop Pipeline
4) Re-start Pipeline
5) Start/Stop Influx, Grafana & Kapacitor
6) Start/Stop Kafka
7) Toggle Pipeline dump logging
8) Truncate Pipeline dump logging
9) Exit
Please enter your choice: 4

!!! Ready to RE-START Pipeline !!!

Press any key to continue or Ctrl+C to exit...

pipeline stop/waiting
pipeline start/running, process 26057
```

6. Close the SSH session to the Telemetry VM.

Section 13: Troubleshooting YANG model data

This section proposes a procedure to validate the data being modelled in YANG or troubleshoot missing values, by checking directly in the XR `SysDB` database.

1. Double click on the XR VM icon to open an SSH session to XRv router under test.



2. Issue the command `show telemetry model-driven sensor-group Sgroup101` and confirm that the sensor paths are resolved. For example, we are going to validate the data for `Cisco-IOS-XR-ip-rib-ipv4-oper`.

```
RP/0/RP0/CPU0:test_XR#show telemetry model-driven sensor-group Sgroup101
Fri Dec 29 00:16:52.170 UTC
Sensor Group Id:Sgroup101
  Sensor Path:      Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
  Sensor Path State: Resolved
  Sensor Path:      Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary
  Sensor Path State: Resolved
  Sensor Path:      Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/data-rate
  Sensor Path State: Resolved
  Sensor Path:      Cisco-IOS-XR-infra-statsd-oper:infra-
statistics/interfaces/interface/latest/generic-counters
  Sensor Path State: Resolved
  Sensor Path:      Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-
names/ip-rib-route-table-name/protocol/bgp/as/information
  Sensor Path State: Resolved
```

NOTE: the sensor path must be in a resolved state, or it is already an early indication that the path specified in the policy is wrong.

3. Issue `show telemetry model-driven sensor-group Sgroup101 internal | begin Cisco-IOS-XR-ip-rib-ipv4-oper` to find for the details (internal keyword) for the `Cisco-IOS-XR-ip-rib-ipv4-oper` sensor-path and note the reported `SysDB` Path.

```
RP/0/RP0/CPU0:test_XR# show telemetry model-driven sensor-group Sgroup101 internal | begin Cisco-IOS-XR-
ip-rib-ipv4-oper
Fri Dec 29 00:21:31.766 UTC
  Sensor Path:      Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-
names/ip-rib-route-table-name/protocol/bgp/as/information
  Sensor Path State: Resolved
  Sysdb Path:      /oper/ipv4-
rib/gl/vrf/<rib_oper_VRF_vrfname>/afi/<rib_oper_AF_afiname>/safi/<rib_oper_SAF_safiname>/table/<rib_oper
IP_RIBRouteTableName_tablename>/proto/bgp/<rib_oper_AS_as>/info
  Yang Path:      Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-
names/ip-rib-route-table-name/protocol/bgp/as/information
```

4. Explore the XR `Sysdb` database (following the instruction in bold the example below, you may use tab for auto completion) by launching the `sysdbcon` application, navigating the database (using `cd` followed by `?` to visualize the options), following the `Sysdb Path` from the previous step and retrieving the path leaf which is the last word in the `Sysdb path` (info in our example).

NOTE: the Sysdb path specifies options fields as < . . . >. You will have to navigate with the cd command to the option and type ? or tab to visualize available options.

```
RP/0/RP0/CPU0:test_XR#run sysdbcon -m
Tue Jan 17 00:32:13.212 UTC
Read-only sysdbcon
Please enter initial bind point>/ <CLICK RETURN FOR />
Enter 'help' or '?' for help
sysdbcon:[m]/> cd ?
  cfg/                [dir]
  ipc/                [dir]
  oper/               [dir]
  debug/              [dir]
  action/             [dir]
sysdbcon:[m]/> cd oper/ipv4-rib/gl/vrf/?
  oper/ipv4-rib/gl/vrf/default/ [dir]de
sysdbcon:[m]/> cd default/afi/IPv4/safi/Unicast/table/default/proto/bgp/65000
sysdbcon:[m]$i/IPv4/safi/Unicast/table/default/proto/bgp/65000/> get info
[bag (len 100)]      'info' (ipv4 rib edm proto v0.1 XDR)
  'ipv4_rib_edm_proto' {
    [string (len 3)]  'ProtocolNames' 'bgp'
    [string (len 5)]  'Instance'      '65000'
    [uint32 (len 4)]  'Version'                0 (0x00000000)
    [uint32 (len 4)]  'RedistributionClientCount' 0 (0x00000000)
    [uint32 (len 4)]  'ProtocolClientsCount'    1 (0x00000001)
    [uint32 (len 4)]  'RoutesCounts'            0 (0x00000000)
    [uint32 (len 4)]  'ActiveRoutesCount'       0 (0x00000000)
    [uint32 (len 4)]  'DeletedRoutesCount'      0 (0x00000000)
    [uint32 (len 4)]  'PathsCount'              0 (0x00000000)
    [uint32 (len 4)]  'ProtocolRouteMemory'     0 (0x00000000)
    [uint32 (len 4)]  'BackupRoutesCount'       0 (0x00000000)
  }
[end bag]
```

5. Double click on “Telemetry VM” icon to log on the Ubuntu server that collects the telemetry data.

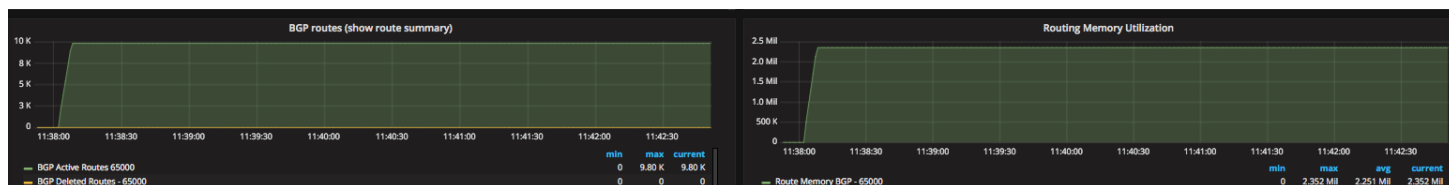


6. Issue the command `/vagrant/inject-bgp.sh` to create the BGP peer and start injecting routes toward the XRv router, wait for the for the **“Updated peers dynamic routes successfully”** to be displayed.

```
vagrant@vagrant-ubuntu-trusty-64:~$ /vagrant/inject-bgp.sh
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | environment file missing
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | generate it using "exabgp --fi >
/usr/local/etc/exabgp/exabgp.env"
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | reactor      | Performing reload of exabgp 3.4.17
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading | neighbor 192.168.10.2 {
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading |     router-id 1.2.3.4;
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading |     local-address
192.168.10.3;
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading |     local-as 65000;
Mon, 16 Jan 2017 06:31:42 | INFO      | 8378 | configuration | loading |     peer-as 65000;
. . .
<SNIP for 2k routes advertised>
. . .
```

7. Back to `sysdbcon` CLI, issue again the `get info` command and confirm the information are consistent with the data streamed by XR and reported in the lab reference dashboard.

```
sysdbcon: [m] $i/IPv4/safi/Unicast/table/default/proto/bgp/65000/> get info
[bag (len 100)]      'info' (ipv4 rib edm proto v0.1 XDR)
  'ipv4 rib edm proto' {
    [string (len 3)]   'ProtocolNames' 'bgp'
    [string (len 5)]   'Instance'      '65000'
    [uint32 (len 4)]   'Version'        0 (0x00000000)
    [uint32 (len 4)]   'RedistributionClientCount' 0 (0x00000000)
    [uint32 (len 4)]   'ProtocolClientsCount' 1 (0x00000001)
    [uint32 (len 4)]   'RoutesCounts'    9801 (0x00002649)
    [uint32 (len 4)]   'ActiveRoutesCount' 9801 (0x00002649)
    [uint32 (len 4)]   'DeletedRoutesCount' 0 (0x00000000)
    [uint32 (len 4)]   'PathsCount'     9801 (0x00002649)
    [uint32 (len 4)]   'ProtocolRouteMemory' 2352240 (0x0023E470)
    [uint32 (len 4)]   'BackupRoutesCount' 0 (0x00000000)
  }
[end bag]
```



8. Exit the `sysdbcon` utility by typing `exit`.

```
sysdbcon: [m] $i/IPv4/safi/Unicast/table/default/proto/bgp/65000/> exit
```

9. Issue the `show route summary` command to validate the consistency with the routes number and memory reported in CLI.

```
RP/0/RP0/CPU0:test_XR#show route summary
Fri Dec 29 02:10:04.502 UTC
Route Source           Routes    Backup    Deleted    Memory(bytes)
local                  4         0         0         960
connected              4         0         0         960
static                 1         0         0         240
dagr                   0         0         0          0
bgp 65000              9801      0         0        2352240
Total                  9810      0         0        2354400
```

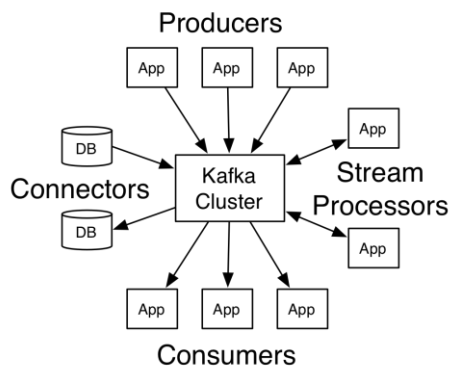
10. Close the SSH session to the Telemetry VM and XRv router.

Section 14: Apache Kafka Pub/Sub bus

As described in <https://kafka.apache.org/>, Apache Kafka® is a distributed streaming platform:

- It lets you publish and subscribe to streams of records. In this respect, it is similar to a message queue or enterprise messaging system.
- It lets you store streams of records in a fault-tolerant way.
- It lets you process streams of records as they occur.

Figure 5. Kafka architecture



In this section, we are going to add Kafka to Pipeline output stages and demonstrate how to create a basic consumer for our streaming telemetry metrics.

1. Double click on “Telemetry VM” icon to log on the Ubuntu server that collects the telemetry data.



2. Launch the `./telemetry_utility.sh` and select option 1 to visualize the current status of the system.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ./telemetry_utility.sh
```

```

1) Verify Environment
2) Destroy & Clean Environment
3) Start/Stop Pipeline
4) Re-start Pipeline
5) Start/Stop Influx, Grafana & Kapacitor
6) Start/Stop Kafka
7) Toggle Pipeline dump logging
8) Truncate Pipeline dump logging
9) Exit
Please enter your choice: 1

```

```
!!! Currently checking for Pipeline, Grafana and Influxdb !!!
```

```
Checking Pipeline ... running
```



```

Input Stage: TCP Dial-Out
Input Stage: gRPC Dial-Out
Input Stage: gRPC Dial-In
Output Stage: TAP dump.txt
Output Stage: Influx TSDB

Checking Grafana ... running

Checking Influxdb ... running

    Pinging Influxdb API ... OK (204)

Checking Kapacitor ... running

    Pinging Kapacitor API ... OK (204)

Checking Kafka ... down

Checking Zookeeper ... down

```

NOTE: Assuming that you followed the lab step by step, you should have at this point Pipeline, Grafana, InfluxDB and Kapacitor running. If you have skipped some sections, please assure that at least Pipeline is running, and it is not, launch Pipeline using option 3.

In this section, we are not going to use InfluxDB, Grafana and Kapacitor. This example, demonstrates that Pipeline supports multiple output stage at the same time. If you want to spare resources in your environment, you can stop these superfluous containers with the `telemetry_utility.sh` option 5, before to continue.

3. Select option 6 (Start/Stop Kafka) from the `telemetry_utility.sh` utility, to start Kafka and update the Pipeline's configuration to export measurements toward Kafka. When requested, use `vagrant` as vault password.

```

Please enter your choice: 6

!!! Ready to START Apache Kafka !!!

Press any key to continue or Ctrl+C to exit...

Vault password:

PLAY [server]
*****

TASK [Retriving secrets]
*****
ok: [server]

TASK [Pre-staging phase - including configuration variables]
*****
ok: [server]

TASK [Checking Pipeline process status]
*****
ok: [server]

TASK [Start Apache Kafka (single broker) and zookeeper]
*****
changed: [server]

```

```

TASK [Pause 10 seconds to let Kafka to start]
*****
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [server]

TASK [Update pipeline.conf for Apache Kafka]
*****
changed: [server]

TASK [Reload Pipeline (init)]
*****
changed: [server]

PLAY RECAP
*****
server                : ok=7    changed=3    unreachable=0    failed=0

```

4. Select option 9 to exit from the `telemetry_utility.sh` utility.

```

Please enter your choice: 9
BYE
vagrant@vagrant-ubuntu-trusty-64:~$

```

5. Issue the command `grep 'Kafka Config' -A 8 ~/environment/pipeline.conf` to visualize the Pipeline configuration output stage to stream to the Kafka bus.

```

vagrant@vagrant-ubuntu-trusty-64:~$ grep 'Kafka Config' -A 8 ~/environment/pipeline.conf
#<!-- BEGIN Ansible Managed - Kafka Config -->
[kafka]
stage = xport_output
type = kafka
# Encoding: gpb, gpbkv, json or json events
encoding = json
brokers = 192.168.10.3:9093
topic = telemetry
# logdata = on
#<!-- END Ansible Managed - Kafka Config -->

```

Notice that the type must be Kafka, it will encode as JSON, the single Kafka broker container answers on port 9093 on the local host and we have specified to use the topic named `telemetry` (to distinguish our data in the bus).

The default TCP Kafka broker port is actually 9092 but it was conflicting with Kapacitor.

6. Kafka and Zookeeper container specifications are defined in the docker compose file `~/environment/kafka-docker/docker-compose-single-broker.yml`, that the `telemetry_utility.sh` calls via the `~/environment/ansible/start_kafka.yml` Ansible playbook.

```

vagrant@vagrant-ubuntu-trusty-64:~$ cat ~/environment/kafka-docker/docker-compose-single-broker.yml
version: '2'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
  kafka:
    build: .

```

```
ports:
  - "9093:9092"
environment:
  KAFKA ADVERTISED HOST NAME: 192.168.10.3
  KAFKA CREATE TOPICS: "telemetry"
  KAFKA ZOOKEEPER CONNECT: zookeeper:2181
volumes:
  - /var/run/docker.sock:/var/run/docker.sock
```

NOTE: The Kafka Docker environment used in this lab, has been contributed as GIT repository by wurstmeister and available for download at <https://github.com/wurstmeister/kafka-docker>. This project provided multiple options to start Kafka as a single used in this example) or more brokers, to horizontally scale and provide redundancy.

Notice that this is the file we have reconfigured to bind the Kafka broker to 9093 instead of the default 9092.

7. Issue `docker ps | grep kafkadocker_kafka` to retrieve the docker ID assigned to the Kafka broker.

```
bash-4.3# docker ps | grep kafkadocker_kafka
cbe23a89c471      kafkadocker kafka      "start-kafka.sh"      About an hour ago    Up About an
hour            0.0.0.0:9093->9092/tcp      kafkadocker_kafka_1
```

8. Log interactively to the Kafka Docker container with the command `docker exec -it <Docker_ID> bash`, by changing the `cbe23a89c471` with the ID retrieved in the previous step.

```
vagrant@vagrant-ubuntu-trusty-64:~$ docker exec -it cbe23a89c471 bash
bash-4.3#
```

9. We can now access Kafka CLI commands and for example query zookeepers for the list of known topics.

```
bash-4.3# kafka-topics.sh --list --zookeeper zookeeper:2181
telemetry
```

NOTE: Remember that we configured `pipeline.conf` to export the streaming telemetry using the `telemetry` topic and we should see this topic reported in the command's output, to confirm Kafka bus is properly processing this stream of data.

10. Use the command `kafka-topics.sh --describe telemetry --zookeeper zookeeper:2181` to obtain more information about the specific topic. Boring in this example with a single basic broker and no replication, important in real configurations with redundancy.

```
bash-4.3# kafka-topics.sh --describe telemetry --zookeeper zookeeper:2181
Topic:telemetry    PartitionCount:1    ReplicationFactor:1    Configs:
  Topic: telemetry    Partition: 0    Leader: 1001    Replicas: 1001Isr: 1001
```

11. Type exit to close the bash process in the Kafka CLI.

```
bash-4.3# exit
exit
vagrant@vagrant-ubuntu-trusty-64:~$
```

12. Issue `cat ~/environment/kafka_examples/kafka_consumer_all.py` to visualize the content of a basic Python consumer script and check the comments.

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat ~/environment/kafka_examples/kafka_consumer_all.py
#!/usr/bin/python2

from kafka import KafkaConsumer          # Import the KafkaConsumer module

# Create a new Kafka consumer and ask for the latest set of data being stored in the Kafka bus.
# Note that you can use auto_offset_reset='earliest' for ask for all the available data.
```

```

consumer = KafkaConsumer(group id='different',bootstrap servers='192.168.10.3:9093',
auto offset reset='latest')

# Suscribe to the telemetry topic, we are exporting from Pipeline
consumer.subscribe(['telemetry'])

# Print every message received for the telemetry topic
for message in consumer:
    print message

```

13. Launch the `python ~/environment/kafka_examples/kafka_consumer_all.py` script and stop it, with CTRL-C command after it starts printing the data received from the messaging bus.

```

vagrant@vagrant-ubuntu-trusty-64:~$ python ~/environment/kafka_examples/kafka_consumer_all.py
ConsumerRecord(topic=u'telemetry', partition=0, offset=21839, timestamp=-1, timestamp_type=0, key='',
value={'Source':"192.168.10.2:42500","Telemetry":{"node_id_str":"test_XR","subscription_id_str":"1","en
coding_path":"Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-
rib-route-table-
name/protocol/bgp/as/information","collection_id":2958808,"collection_start_time":1514520669859,"msg_tim
estamp":1514520669859,"collection_end_time":1514520669912},"Rows":[{"Timestamp":1514520669896,"Keys":{"a
f-name":"IPv4","as":"65000","route-table-name":"default","saf-name":"Unicast","vrf-
name":"default"},"Content":{"active-routes-count":9801,"backup-routes-count":0,"deleted-routes-
count":0,"instance":"65000","paths-count":9801,"protocol-clients-count":1,"protocol-
names":"bgp","protocol-route-memory":2352240,"redistribution-client-count":0,"routes-
counts":9801,"version":0}}]}', checksum=1390084074, serialized key size=0, serialized value size=797)
<SPIN>

```

NOTE: This script is not very interesting because it just prints the JSON messages produced by Pipeline but it provides the basic blocks to create a Python Kafka consumer and we can look in the actual JSON data to create something more interesting.

14. Issue `cat ~/environment/kafka_examples/kafka_consumer_filter.py` to visualize the content of a slightly more interesting Python consumer script.

```

vagrant@vagrant-ubuntu-trusty-64:~$ cat ~/environment/kafka_examples/kafka_consumer_filter.py
#!/usr/bin/python2

from kafka import KafkaConsumer
import json                                     # import json module to process JSON messages

consumer = KafkaConsumer(group id='different',bootstrap servers='192.168.10.3:9093',
auto offset reset='latest')
consumer.subscribe(['telemetry'])

# In this example we are going to filter on specific measurements and alram on KPIs
host ip="192.168.10.2"
measurement="Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization"
kpi=15

# This basic loop processes every message received from the Kafka Bus and triggers a notification
# when the total-cpu-one-minute is more that 15%
for message in consumer:
    data=json.loads(message.value)
    if data["Telemetry"]["encoding_path"] == measurement:
        if data["Rows"][0]["Content"]["total-cpu-one-minute"] > kpi:
            print("Trigger: one minute CPU on %s is %i" %( \
                data["Telemetry"]["node id str"], \
                data["Rows"][0]["Content"]["total-cpu-one-minute"]))

```

NOTE: As you can read in the code comments, this script checks the incoming JSON message (from the Kafka's subscription) and triggers a notification when total-cpu-one-minute exceeds 15%. Think this example as the poor man version of the CPU KPI, we previously implemented in Kapacitor.

15. Launch the `python ~/environment/kafka_examples/kafka_consumer_filter.py` script.

```
vagrant@vagrant-ubuntu-trusty-64:~$ python ~/environment/kafka_examples/kafka_consumer_filter.py
```

The script consumes Kafka notification but we may need to increase the CPU utilization on the XRv, before seeing any alert.

16. Double click on "Telemetry VM" icon to create new SSH session to the Ubuntu server that collects the telemetry data.



17. Type `sudo ping 192.168.10.2 -s 10000 -f` in the new CLI, to initiate a flood ping from the Linux server to the XRv router and increase the average CPU utilization.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo ping 192.168.10.2 -s 10000 -f
PING 192.168.10.2 (192.168.10.2) 10000(10028) bytes of data.
.^
--- 192.168.10.2 ping statistics ---
2376 packets transmitted, 2376 received, 0% packet loss, time 5209ms
```

18. Check back on the running `kafka_consumer_filter.py` script. It should start triggering exception when the average CPU utilization, on the one minute sliding window, reaches 15%. Remember that you can track it on the lab reference dashboard that we have explored in [section 8](#).

```
vagrant@vagrant-ubuntu-trusty-64:~$ python ~/environment/kafka_examples/kafka_consumer_filter.py

Trigger: one minute CPU on test XR is 20
Trigger: one minute CPU on test XR is 20
Trigger: one minute CPU on test XR is 20
Trigger: one minute CPU on test_XR is 20
```

NOTE: Notice that it may takes few seconds for the CPU to increase because measuring CPU average on 30 seconds sliding window.

19. Issue a CTRL-C command to stop the continuous ping and a second CTRL-C command to stop

```
kafka_consumer_filter.py.
```

20. You can now close the SSH sessions.

Section 15: Cleaning up

1. Double click on "Telemetry VM" icon to log on the Ubuntu server that collects the telemetry data.



2. Type `./telemetry_utility.sh` to launch the support script, select option 2 to clean up the environment, using `vagrant` as vault password.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ./telemetry_utility.sh

1) Verify Environment
2) Destroy & Clean Environment
3) Start/Stop Pipeline
4) Re-start Pipeline
5) Start/Stop Influx, Grafana & Kapacitor
6) Start/Stop Kafka
7) Toggle Pipeline dump logging
8) Truncate Pipeline dump logging
9) Exit
Please enter your choice: 2

!!! Ready to shutdown Pipeline, destroy all Docker containers and clean timeseries, dumps and log !!!

Press any key to continue or Ctrl+C to exit...
Vault password:

PLAY [server]
*****

TASK [Retriving secrets]
*****
ok: [server]

TASK [Pre-staging phase - including configuration variables]
*****
ok: [server]

TASK [Check if Pipeline service is configured]
*****
ok: [server]

TASK [Stop Pipeline process ( doesn't stop Pipeline if manually launched)]
*****
changed: [server]

TASK [Shutdown and delete Influx, Grafana & Kapacitor containers]
*****
changed: [server]

TASK [Shutdown and delete Kafka (single broker) and zookeeper]
*****
changed: [server]
```

```
TASK [Clean data directory]
*****
changed: [server]

TASK [Clean log directory]
*****
changed: [server]

TASK [Remove pipeline.conf from upstart configuration (/etc/init/)]
*****
changed: [server]

TASK [Remove Influx consumer from pipeline.conf]
*****
changed: [server]

TASK [Remove Kafka producer from pipeline.conf]
*****
changed: [server]

PLAY RECAP
*****
server                : ok=11   changed=8    unreachable=0    failed=0
```

Section 16: What Next

If you are interested running or customize this lab in your laptop or stage a server in your lab by reusing the software packaged for this demonstration, you should check the public Github repository [telemetry-staging-ansible](#).

When planning for a local installation, you may consider one of the following options:

1. Clone this project repository, including the Telemetry VM Vagrant image in your laptop or server.
 - Easier option, perfect for live demonstration, start in few minutes and doesn't require Internet connectivity (after downloading the repository and images).
2. Clone this project repository and stage the Telemetry Server from an empty Ubuntu image with the provided Ansible playbook.
 - Provide maximum flexibility, you can update the software version downloaded when installing and you can still follow this documentation.
 - The final environment still includes the XRv router and the Telemetry VM.
3. Clone this project and stage using Ansible, a separate VM or baremetal in your lab.
 - Best option if you are planning for a POC and you want to reuse the configuration proposed for the Telemetry VM.

If you are interested in any of these options, please follow the instructions in the project [README](#) file.



Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)