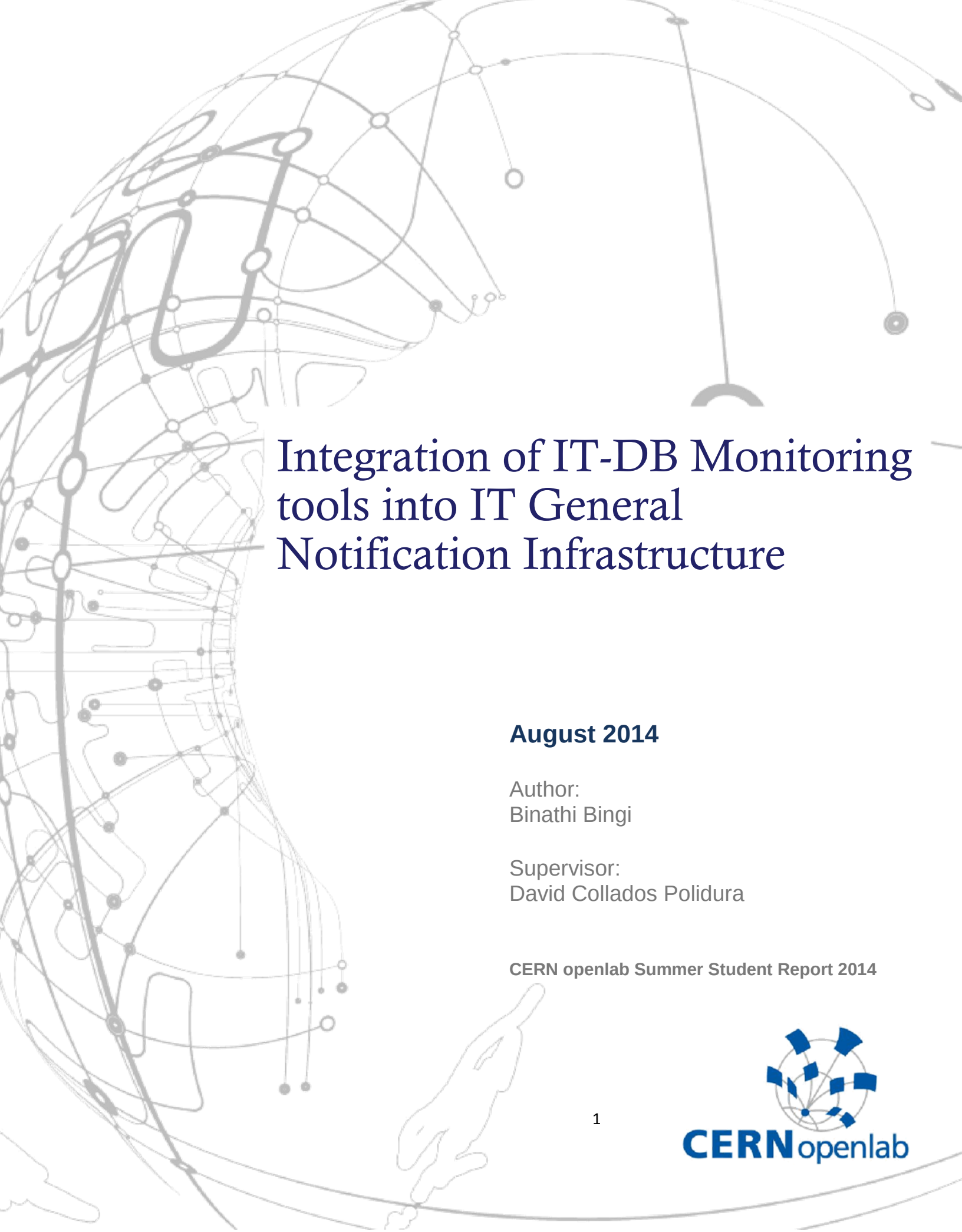


Integration of IT-DB Monitoring tools into IT General Notification Infrastructure

August 2014

Author:
Binathi Bingi

Supervisor:
David Collados Polidura

CERN openlab Summer Student Report 2014

CERN openlab

Project Specification

The goal of this openlab summer student project was to standardize the service notification and alarming system in the IT Database group. For this we need to integrate the IT General Notification Infrastructure (GNI) into some of our database services, like for instance, RACMon, Enterprise Manager, Syscontrol, RMAN, Database on Demand, and Storage Administrators' tools. The objective was to make the GNI service our only mechanism to generate notifications and alarms (SMS, email, SNOW tickets) and as unique interface to visualize notifications.

Abstract

The IT Database group has independent monitoring tools/data and is immersed in a process of consolidating its monitoring infrastructure. The aim of this document is to provide insight into the way we achieved integration of GNI into our database services.

Table of Contents

1 Introduction

The IT Monitoring team is working to provide monitoring solutions in different areas.

Notifications: To get notified about problems affecting our services or nodes.

Archive: To archive data for offline batch analysis or historical reference.

Dashboards: To visualize monitoring data based on real time analysis.

To deliver successful monitoring solutions it is important to continuously monitor the status of all resources (network equipment, physical machines, virtual machines, operating systems, application services, etc.), to efficiently process all collected data, to promptly deliver monitoring results (notifications, alarms, reports, etc.) to the appropriate target, and to have the capability of executing complex queries across distinct monitoring data sets.

The IT-Database (IT-DB) group is in a process of consolidating the entire monitoring infrastructure. As a part of it we created a single mechanism (DB_notifier Producer) to generate notifications and alarms. We adopted a single interface Kibana, to visualize the monitoring data. Our goal was to stop sending notifications in the form of emails as much as possible and automate the opening of SNOW (Service Now) tickets for certain critical events.

1.1 Database monitoring tools

In IT-DB there are independent monitoring tools like Oracle Enterprise Manager (OEM), Syscontrol, RACMon, Lightweight monitoring tools, monitoring data/scripts for middleware tools like Syscontrol, or for other services like Storage, Backups, Recoveries, etc. These multiple monitoring applications are deployed for monitoring the status of specific resources. These were independent tools based on different tool chains. Despite their heterogeneity, they all shared a similar architecture and faced the same limitations, leading to unnecessary duplication of effort and increased difficulties in sharing monitoring data.

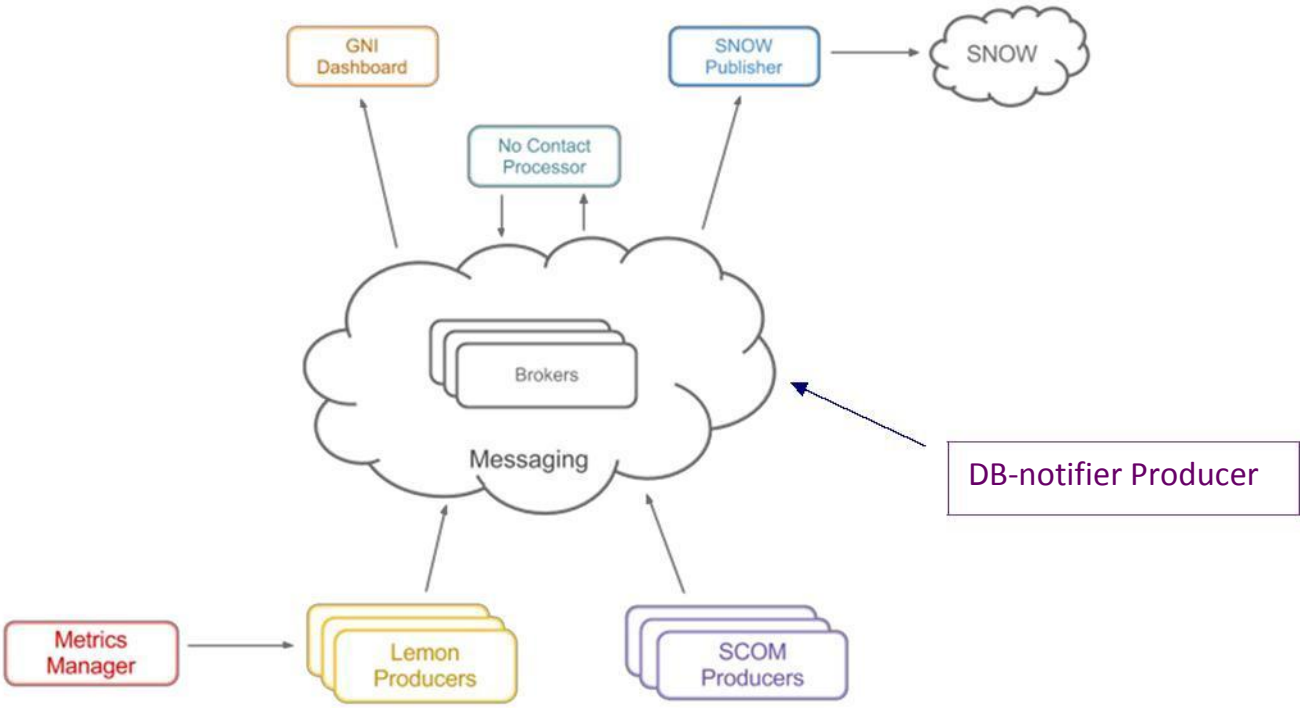
1.2 GNI and SNOW

GNI is a notification layer composed of several components responsible for dispatching alarms triggered in each data centre node to multiple operations tools. Different from other tools, GNI relies on dedicated transport layer, based on messaging brokers. Different producers publish notifications to the messaging infrastructure, which are taken by two consumers: one which creates tickets (SNOW tickets) in the CERN central ticketing system [1], while the second consumer populates a web application (Kibana) [2] showing current notifications.

1.3 Kibana

Kibana is a browser based analytics and search interface for Elasticsearch that was developed primarily to view Logstash event data. It is used by IT monitoring to visualize monitoring data in real time. Kibana is user friendly dashboard and can be configured to display data as per the user requirements. It facilitates query on the monitoring data and also to save the dashboards.

2. Consolidation of IT-DB monitoring



In the process of consolidation of monitoring tools, we created a DB-notifier producer which would take messages in a specific format as input from the monitoring tools to generate notifications. These notifications are sent to the messaging broker in the cloud. Further, the notifications are forwarded to the consumers like GNI and SNOW depending on the type of notification.

2.1 Input

DB_notifier Producer takes two arguments as input.
Messages: In the form of key value pair.
Configuration file: configuration files of monitoring tools generating the message.

Messages from Syscontrol monitoring tool are in the form of key value pair as shown below:
Feb 12 16:15:37 itrac1111 SYSCONTROL_LOGGER[23286]: |V:002|BEGIN_REMOTE:17837630655748|USER_LOCAL:sysctl|CMD:ssh|USER_REMOTE:sysctl|HOST:dbsrvg3301|SCRIPT:/etc/init.d/syscontrol|SINGLE:single|SUSPEND:|RUN_AS_RE

MOTE:|IGNORE_ACTIVE:|TAG:tagenable_printers|DELAY:|DELAY_VAL:|SILENT:very_sil
ent|HOST:dbsrvg3301|ENTITY:enable_printers|OPS:start|ARGS:|APPLY_PROFILE:|

DB_notifier Producer facilitates an option to provide either individual message or a log file of messages as input. Monitoring tools like OEM, RACMon generate messages in text format. We wrote a shell script that would consume these text messages and generate the message in required key value pair format. Whenever there is an event triggered in the monitoring tool, it would make a call to the shell script. The script would consume the text message, generate output in proper format and pass it as input to the DB_notifier producer.

The second argument passed to DB_notifier is the configuration file of the monitoring tool. We had written configuration files for each monitoring tool in IT-DB, these are passed as argument to the DB_notifier producer.

Example: Configuration file of the Syscontrol monitoring tool

```
[configuration]
destination_dir = tmp/teststomp

[header]
Version        =  2.0
Type           =  notification
des            =  /topic/monitoring.notification.db
env            =  dbnotifier_dev
host_group     =  physicsdb/pdbbackup
producer       =  dbnotifier_syscontrol
snow           =  1
```

2.2 DB_notifier Producer

DB_notifier producer is a python script that takes the configuration file of the monitoring tool and output of the shell script as input to generate a JSON(JavaScript Object Notation) file of the notifications. The notification specification [3] given by IT Monitoring must be followed in order for the GNI to accept the notification.

Message structure:

```
{

  "header":{ ... },

  "body":"{

    "payload": { ... },

    "metadata": { ... }

  }"

}
```

Notification specification mandates the inclusion of certain fields in the header and body of the messages as specified below. Current version of notification specification is v2.0

Header tags:

TAG	MANDATORY	TYPE	DESCRIPTION
m_version	yes	string	version of message specification
m_type	yes	string	type of message
m_producer	yes	string	identifier of the application producing messages
m_submitter_environment	yes	string	environment of the submitter host
m_submitter_hostgroup	yes	string	cluster of the submitter host
m_submitter_host	yes	string	submitter host
m_toplevel_hostgroup	no	string	top level hostgroup of the source host
m_snow	no	string	enable or disable snow tickets
destination	no	string	destination of the message (necessary for the stompclt)

Body/Metadata tags:

TAG	MANDATORY	TYPE	DESCRIPTION
timestamp	yes	int	time when the event occurred (seconds since epoch)
uuid	yes	string	unique message identifier
metric_id	yes	int	metric id
metric_name	yes	string	metric name
entity	yes	string	source host
hostgroup	no	string	hostgroup of the source host (snow defaults to NO_HOSTGROUP)
environment	no	string	environment of the source host
is_essential	no	string	is_essential flag of the source host (snow defaults to "0")
asset_id	no	string	identifier of the source machine
description	no	string	detailed description of the notification, used in the SNOW title of the INC (defaults to "no description provided" in snow)
notification_type	no	string	type of the notification (os, app, hw, nc)

state	no	string	local state of the notification: open, close, *
validity	no	int	notification validity in hours (default 24h)
egroup_name	no	string	egroup responsible for entity
fe_name	no	string	name of functional element to send snow ticket
troubleshooting	no	string	url or comment for troubleshooting information
snow_assignment_level	no	int	assigment level to created Snow ticket, defaults to 2
snow_grouping	no	string	boolean to request or not grouping of ticket in snow based on hostgroup and metric_name over a 1hour period, defaults to "1"
snow_instance	no	string	the CERN snow instance: cern, cerndev, cerntest, cernsandox
snow_display_value	no	string	the incident number of the record in snow
snow_id	no	string	sys_id of already created snow ticket (event record)

Body/Payload tags:

TAG	MANDATORY	TYPE	DESCRIPTION
*	no	*	*

General format for a custom notification producer:

```
#/usr/bin/python
from monitoringdatamodel.gni import Notification
from monitoringdatamodel import utils
from messaging.message import Message
from messaging.queue.dqs import DQS
import time
import socket

if __name__ == "__main__":
    # Local path to send message so stomp client can consume it
```

```
mq = DQS(path="/tmp/test")
# You can see the complete specification of a notification in:
my_notification_header = {
   'm_type': 'notification',
   'm_version': '2.0',
   'm_producer': 'mycustomscript',
   'm_submitter_environment': 'qa',
   'm_submitter_hostgroup': 'aimon/my/host/group',
   'm_submitter_host': socket.gethostname(),
   'destination': '/topic/monitoring.notification.generic'
}
my_payload = {
   "MyValues": [1, 2, 3],
   "custom": "field"
}
my_notification_body = {
   'payload': my_payload,
   'metadata': {
      'metric_id': 123456,
      'metric_name': 'my_big_problem_alarm',
      'entity': 'my_little_machine',
      'timestamp': int(time.time()),
      'destination': '/topic/monitoring.lemon.notification',
      'uuid': utils.generate_uuid(),
   }
}
notif = Notification({
   'header': my_notification_header,
   'body': my_notification_body
})
# We send the message to the queue directory
mq.add_message(notif.to_message())
```

The above producer has certain dependencies:

Monitoring-data-model: to build valid notifications[4].

python-messaging: for writing in the notifications queue of the broker[5].

stompclt: necessary dependency to send message to the messaging broker[6].

These dependencies are Red Hat Linux6 dependent rpms and can be installed using yum.

```
yes | yum install stompclt monitoring-data-model python-messaging
```

Now, we have a single notification producer for all the monitoring tools/data in IT-DB. The output of the notifier is sent to the messaging brokers using client that uses STOMP protocol.

2.3 Messaging Broker

stompclt is a versatile tool to interact with messaging brokers speaking STOMP and/or message queues on disk. It receives messages from an incoming module, optionally massaging them (i.e. filtering and/or modifying), and sends them to an outgoing module. Depending on which modules are used, the tool can perform different operations.

Here are the supported incoming modules:

- broker: connect to a messaging broker using STOMP, subscribe to one or more destinations and receive the messages sent by the broker
- queue: read messages from a message queue on disk

Here are the supported outgoing modules:

- broker: connect to a messaging broker using STOMP and send the messages
- queue: store the messages in a message queue on disk

In our case, stompclt is the messaging broker that forwards the JSON files from the DB_notifier Producer to the GNI. stompclt must be configured to the incoming queue path and is done in the stompclt-notifications.conf file.

Certain credentials are needed to access stompclt. They are configured in the stompclt-auth.conf file.

2.4 Output

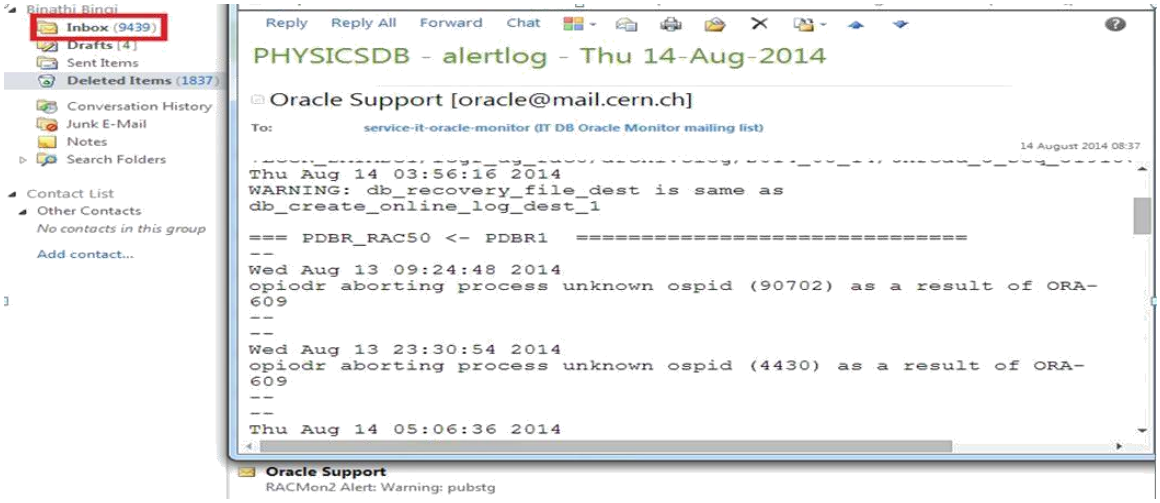
JSON format files are forwarded from the messaging brokers to the GNI Dashboards by a client.

Now, we can visualize the notification data in real time and perform query on it. In addition, generation of automatic SNOW tickets is facilitated for critical events.

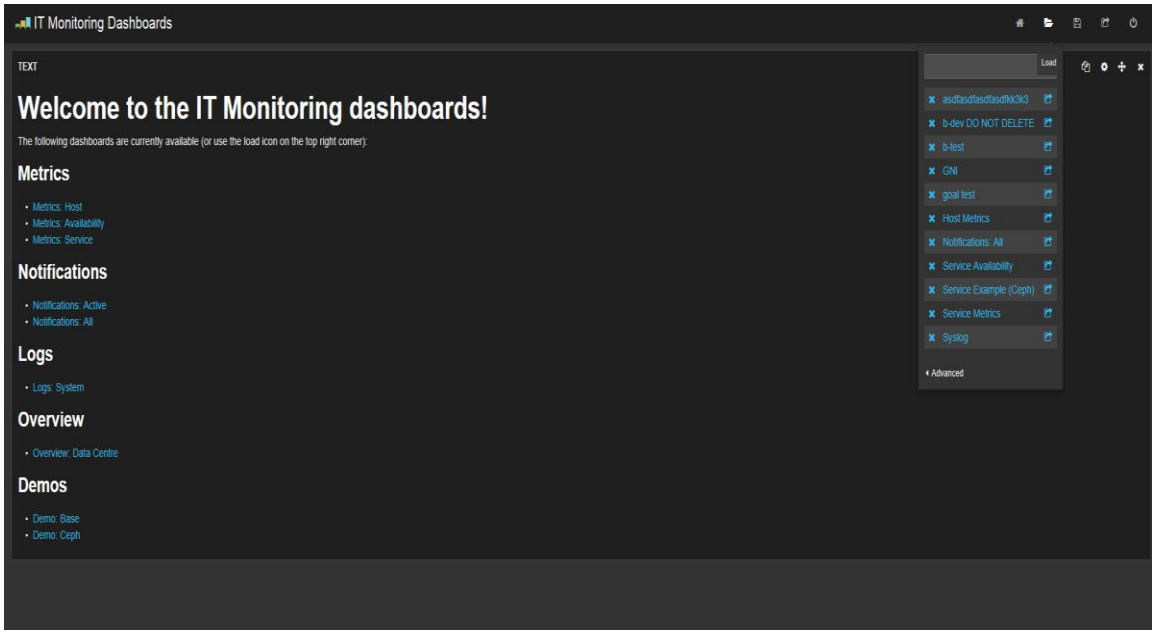
3. Visualizations

Kibana interface is used for visualization of data. We can login into dashboard to see, create and query on the monitoring data generated by IT-DB production monitoring tools.

The previous notification system was in the form of emails. In November 2013, we received almost 6638 emails, approximately 221 emails per day. It was very difficult to share or query the monitoring data using the previous notification system.

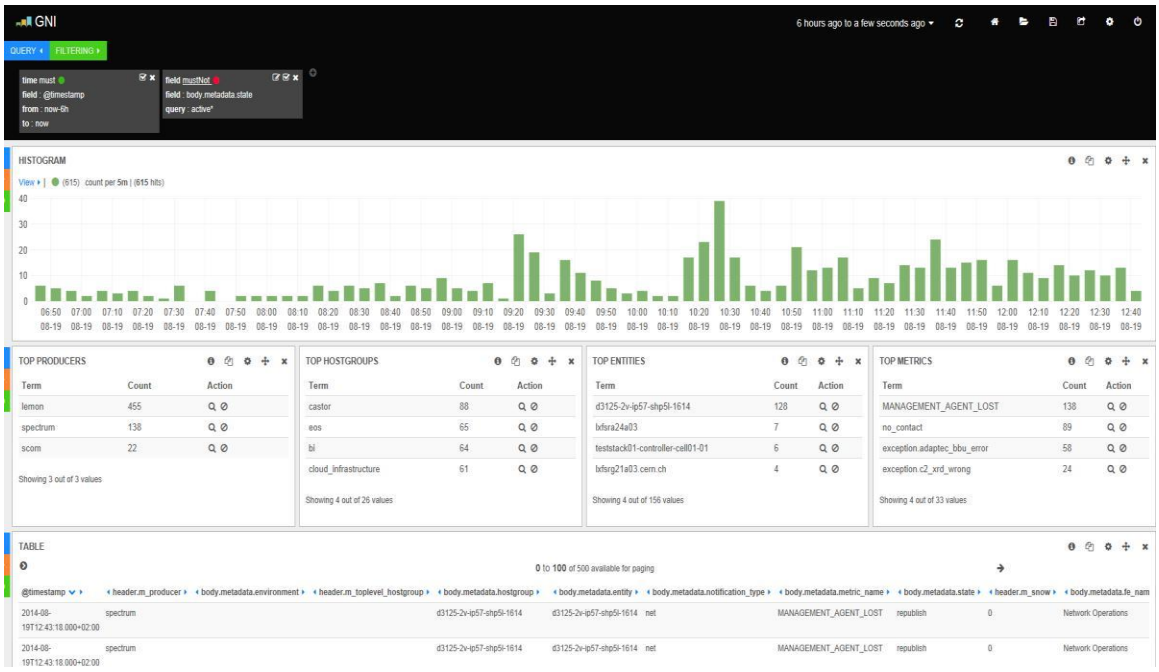


DB_notifier Producer has facilitated us to make use of the dashboards to visualize notifications rather than receiving them in form of emails.

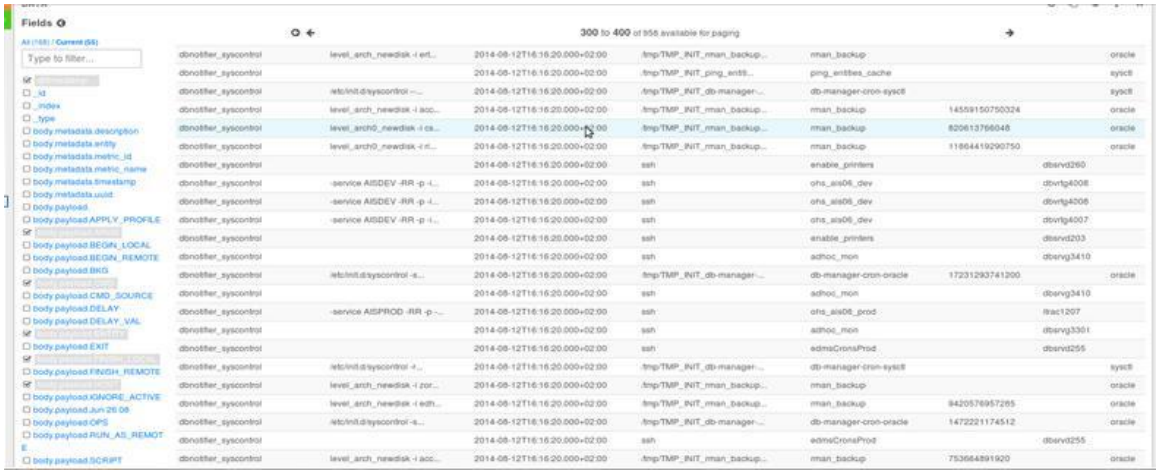


Kibana interface

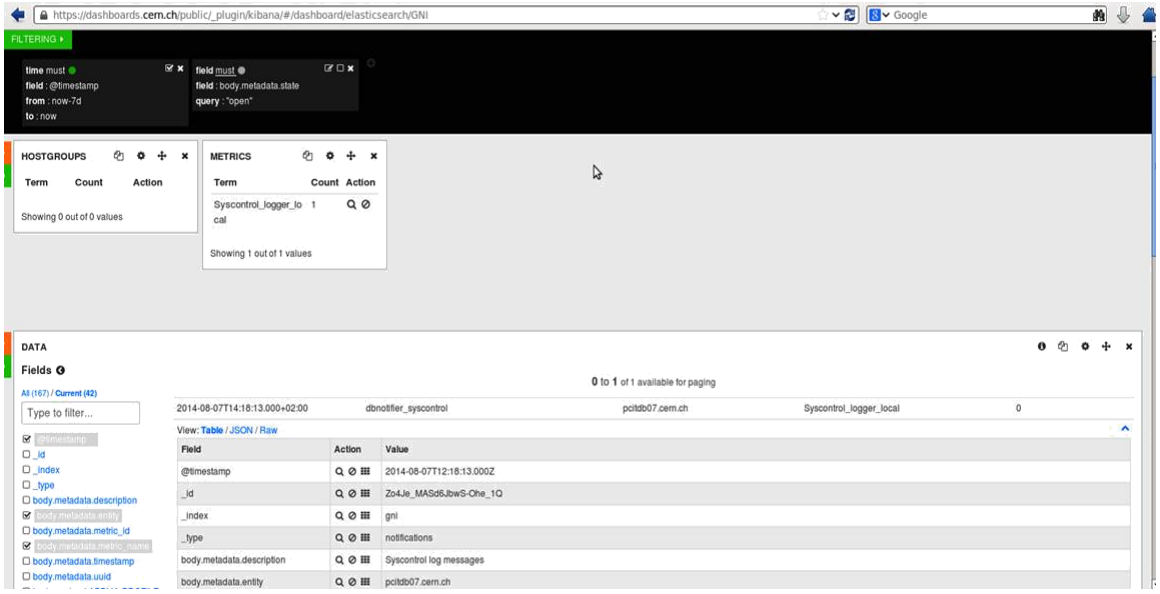
Many dashboards are available on Kibana. GNI dashboard can be loaded onto the Kibana.



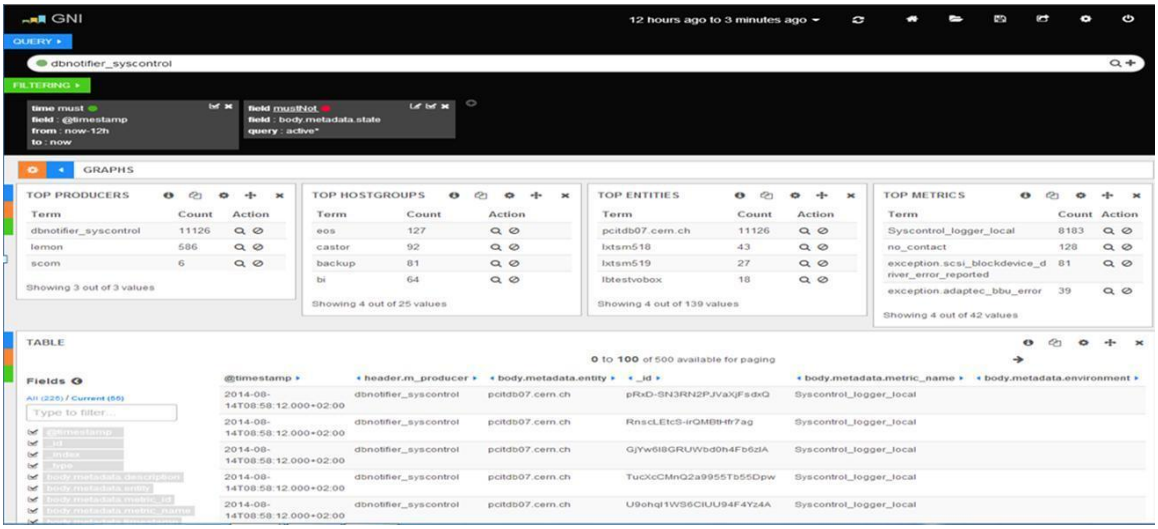
GNI Dashboard



Dashboards also display various fields regarding each event. It is possible to choose the fields to be displayed for each event and query using the value of the fields.



We can also query for top host groups or producers, etc. as per the requirement and configure how the result of our query would be displayed (either in the form of tables, or graphs like histograms, pie charts, etc.).



For example if we run a query for top producers on GNI dashboard and configure to display result in the form of a pie chart. Clicking on any of the sectors of the pie chart would display the concerned notification data.

4. Conclusion

Through this project we achieved

Single IT-DB producer: DB_notifier Producer that we developed is the single notification producer for all the IT monitoring tools in IT-Databases. It is easy to maintain and run same notifier in all the production monitoring machines.

Visualize and query monitoring data: Display of data on Kibana interface facilitates visualization of data in the form of graphs and tables. It is easy to query or filter the monitoring data using the tools available on Kibana.

Configure and display monitoring data: Dashboards can be configured to display the monitoring data that interests the user. These dashboards can be saved and then shared, imported or exported.

Email notifications: We could stop global email notifications to a considerable level by putting DB_notifier Producer into production. Events, alerts, etc. are displayed on the dashboards rather than sending them in the form of emails to the group users.

SNOW tickets: Automatic Service NOW tickets can be opened for events or alerts that are critical enough.

5. Future Proposals

At present, production monitoring machines (db-manager, oem.cern.ch, etc.) run on RedHat Linux5. DB_notifier Producer that we developed has RedHat Linux6 dependencies. Servers need to be migrated to RedHat Liux6 in order to bring DB_notifier Producer into production. We also need to identify the messages that should generate SNOW tickets and put it into production. Currently we integrated Syscontrol, OEM into GNI and are working on RACMon. We can further add more IT-DB monitoring tools into GNI.

6. Bibliography

1. https://cern.service-now.com/service-portal/
2. https://dashboards.cern.ch/public/_plugin/kibana/#/dashboard/elasticsearch/Notifications:%20All
3. https://itmon.web.cern.ch/content/build-custom-notifications-consumer
4. http://itmon-doc.web.cern.ch/itmon-doc/monitoring-data-model/
5. https://messaging.readthedocs.org/en/latest/index.html
6. http://mig.web.cern.ch/mig/doc/stompclt.html