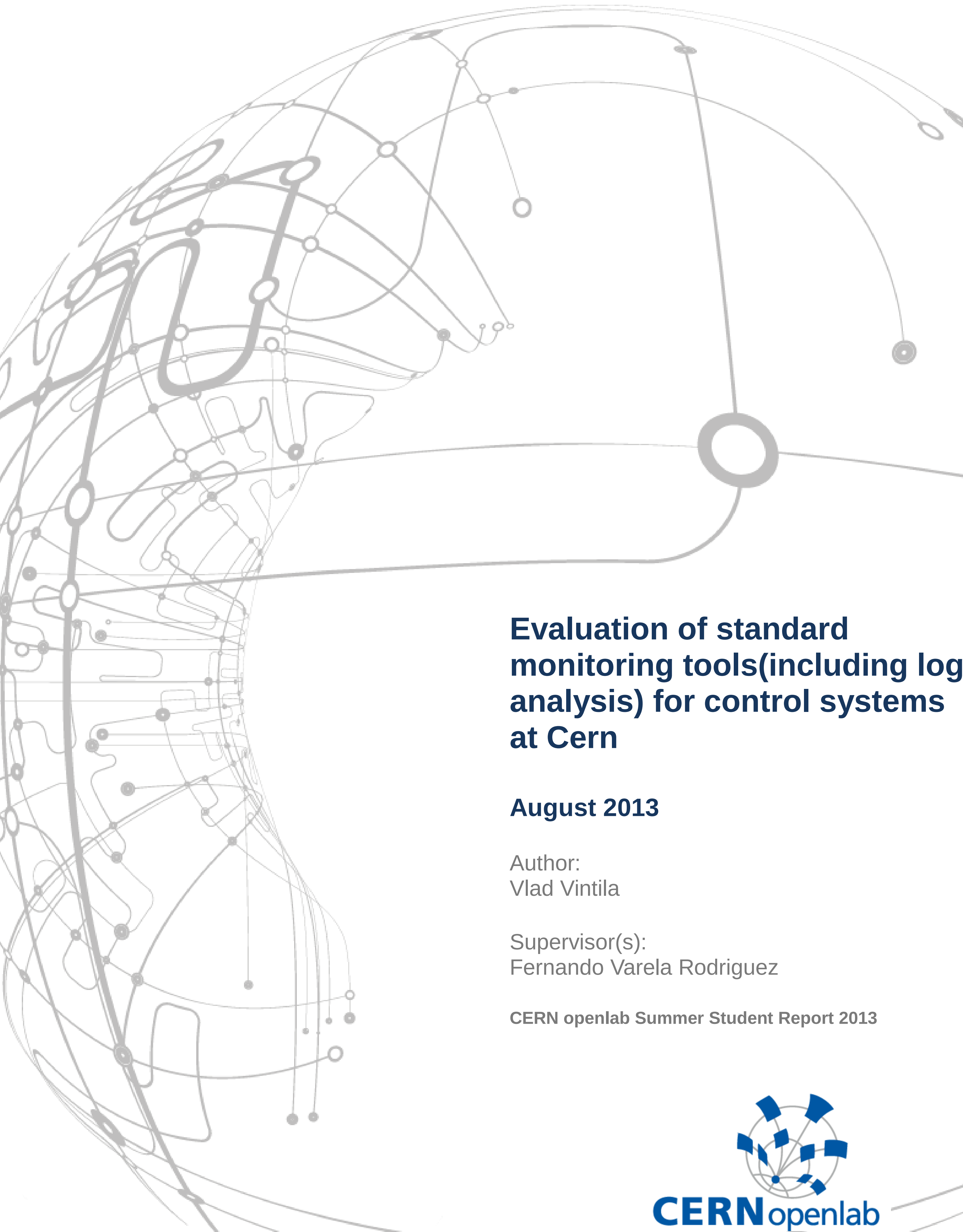


Evaluation of standard monitoring tools(including log analysis) for control systems at Cern

August 2013

Author:
Vlad Vintila

Supervisor(s):
Fernando Varela Rodriguez

CERN openlab Summer Student Report 2013

Project Specification

The goal of this Openlab Summer Student project was to assess the implications and the benefits of integrating two standard IT tools, namely Icinga and Splunkstorm with the existing production setup for monitoring and management of control systems at CERN.

Icinga – an open source monitoring software based on Nagios would need to be integrated with an in-house developed WinCC OA application called MOON, that is currently used for monitoring and managing all the components that make up the control systems.

Splunkstorm – a data analysis and log management online application would be used stand alone, so it didn’t need integration with other software, only understanding of features and installation procedure.

Abstract

The aim of this document is to provide insights into installation procedures, key features and functionality and projected implementation effort of Icinga and Splunkstorm IT tools. Focus will be on presenting the most feasible implementation paths that surfaced once both software were well understood.

Table of Contents

1 Introduction

Two key components of the existing monitor and management system in place for industrial controls needed to be looked at, namely the monitoring of servers' health, processes, parameters and log centralization for all log producing entities in the control systems setups.

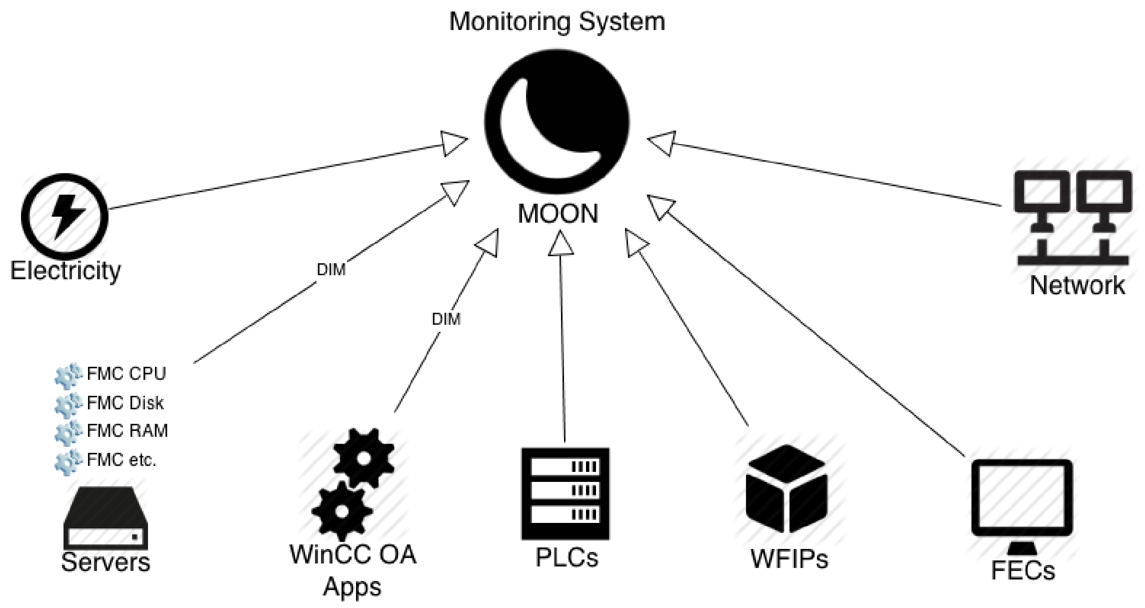


Figure 1. Current setup

For montoring the standard setup (*Figure 1.*) is currently as follows: FMC(Farm Monitoring and Control) agents running on each server that monitor cpu, disk, memory, processes health etc. These agents do specialized tasks, so there is a different one for each different check that needs to be done. They act as DIM (Distributed Information Management System – a CERN developed protocol) servers to which the monitoring system, called MOON connects to obtain the raw data that each agent is producing.

This was a proven, robust system. However the problem with this is that it takes a lot of time to maintain the existing FMC agents and to develop new ones. Being C or C++ programs written by CERN personnel, they do exactly what they need to do and were surely the right choice to go with at the time of their introduction. With years going by, open source projects aiming to provide monitoring tools began to appear and mature. Also, with more and more companies developing complex computer infrastructures, CERN's needs stopped standing out as special and quickly became standard. Taking all these into account, it was a logical step to look into the replacement of FMC agents with one of the 'standard' open source solutions.

For logging there is no current solution in place. Taking into account that being able to visualize and analyse log messages in a centralized system or tool is a proven, powerful way to debug flaws in software applications and to single out the causes of a failure it was mandatory that a centralized log analysis solution would be looked at.

We will be looking at Icinga and Splunkstorm one by one and see the discoveries made by studying the implementation of these two.

2 Icinga

2.1 Overview

Icinga is an open source network and computer system monitoring application. It was originally created as a fork of the Nagios monitoring application in 2009. Icinga tries to address the shortcomings of Nagios by adding functionality and trying to create more ways of integrating with other software, all while keeping configuration and plugin compatibility with Nagios.

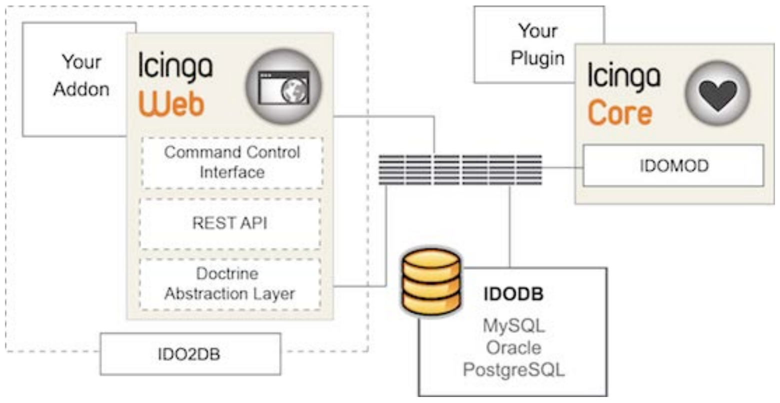


Figure 2. Icinga architecture

Building on top of a proven monitoring tool, with the goal of being more easily extendable and easier to integrate, Icinga was the choice that stood out of the crowd, so it was chosen for evaluation.

Overly simplified, Icinga just runs scripts and checks the exit status, saves that and the provided output

Ex:

```
value = example_probe_metric()

if value >= critical_threshold:
  print "CRITICAL – Value is bigger than %d" % critical_threshold
  sys.exit(2)

elif value >= warning_threshold:
  print "WARNING - Value is bigger than %d" % warning_threshold
  sys.exit(1)

elif
  print "OK – All ok"
  sys.exit(0)
```

The exit status code meaning is:

0 - OK

1 – WARNING

2 - CRITICAL

3 - UNKNOWN

Using the same logic, to run checks remote scripts (plugins) on other hosts the check_nrpe script is used. It sends a command to the listening NRPE daemon on the monitored server, the script for that particular command is run and the output and exit status are forwarded to Icinga. See *Figure 3.*

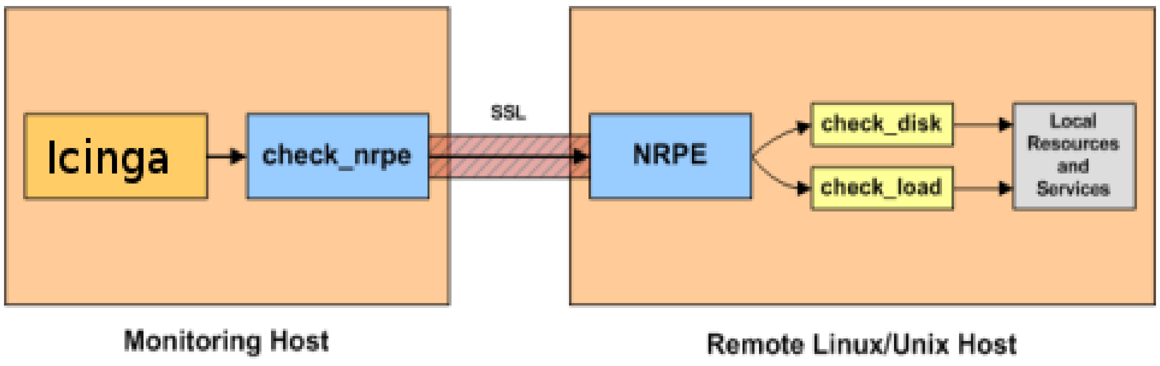


Figure 3. Execution model

2.2 Features and advantages

The advantages of replacing FMC agents with Icinga consisted of shifting the development and maintenance efforts from CERN employees to the community around the product, thus enabling them to concentrate on other areas. There is also potential for added functionality over the FMC setup. We're going to look through some of the features that are appealing and relevant to the EN-ICE-SCD section.

As we can see from *Figure 2.* it is not a monolithic application, but rather a set of components doing specific jobs. This allows for flexibility in the setup Icinga requires for installation and also aids scalability quite a lot. For example you can have multiple icinga-core instances running on different servers using the same database.

Another advantage of this structure is that a large number of communication points can be opened. You can access Icinga data directly from the core, using the Event Broker API, use the REST API, access the database, etc.

Full compatibility with nagios-plugins means that Icinga benefits from the large ecosystem of existing nagios plugins to monitor a wide variety of metrics and components and also from the simplicity of the structure of nagios monitoring scripts. To create a plugin only an exit status and a output is required. All you need to focus on is the logic behind obtaining the value or values for what a particular check.

Another aspect that is worth mentioning as an advantage is the small effort required in understanding and building the configuration for monitoring your setup. Here is an example of a service(check) definition:

```
define service{
        host_name                linux-server
        service_description      check-disk-sda1
        check_command            check-disk!/dev/sda1
        use                      generic-service
        }
```

One interesting here to note here is the "use" directive, which means that our service inherits different settings from a template, called generic-service, thus removing clutter and speeding up configuration time.

2.3 Installation

We are going to briefly explain the installation procedure of an Icinga setup. For a full command by command how to and config file details please see Annex A – Installation of Icinga

Icinga server:

- Install oracle compatibility on the server, including ocilib, which has to be compiled
- Install icinga package from rpmforge repository
- Install idoutils – needed for writing data in the database - by downloading icinga binary and compiling only idoutils. This must be done because the idoutils package from rpmforge doesn't include oracle support
- Configure both icinga and idoutils and start both
- Create the schema needed for icinga on the database
- Install nagios-plugins and relevant individual nagios plugins to be used in monitoring (ex: nagios-plugins-ssh, nagios-plugins-disk, etc.)

Monitored server:

For LINUX:

- Install nagios-nrpe from rpmforge repository
- Install nagios-plugins (see above)
- Configure and start
- Open port 5666 tcp(this is what the nrpe daemon binds on)

For WINDOWS:

- Install NSClient++
- Configure and start

2.4 Integration

After installing and understanding Icinga, the best way to integrate it with the current setup soon stood out. This was using a plugin which uses the Event Broker API to talk directly with the icinga-core, and providing a simplified but powerful API available for querying, providing control on the output type of the information. This is to be used together with a WinCC OA program that queries this plugin to get the data out of Icinga and in to moon.

This plugin is called Livestatus. For details on installing it, please see http://docs.icinga.org/latest/en/int-mklivestatus.html and Annex A. One thing to note is that by default Livestatus binds to a Unix socket, so in order to make it network accessible is to use a xinetd configuration to create a "wrapper" that redirects traffic from a tcp socket to the Unix socket. You can find the configuration in Annex B – Xinetd livestatus configuration.

Advantages of Livestatus include performance, full functionality and the fact that it doesn't need a database to operate. For full features and functionality, see http://docs.icinga.org/latest/en/int-mklivestatus.html.

Two scripts were created to showcase the functionality of Livestatus. The first one (Annex C) connects to the UNIX socket and does a query. One thing to note is that this can be run only on the same host. The second (Annex D), a bit more complex, queries the tcp socket provided by xinetd, and also formats the output. This can be run from anywhere as long as it has connectivity with the icinga server on the desired port.

3 Splunkstorm

3.1 Overview

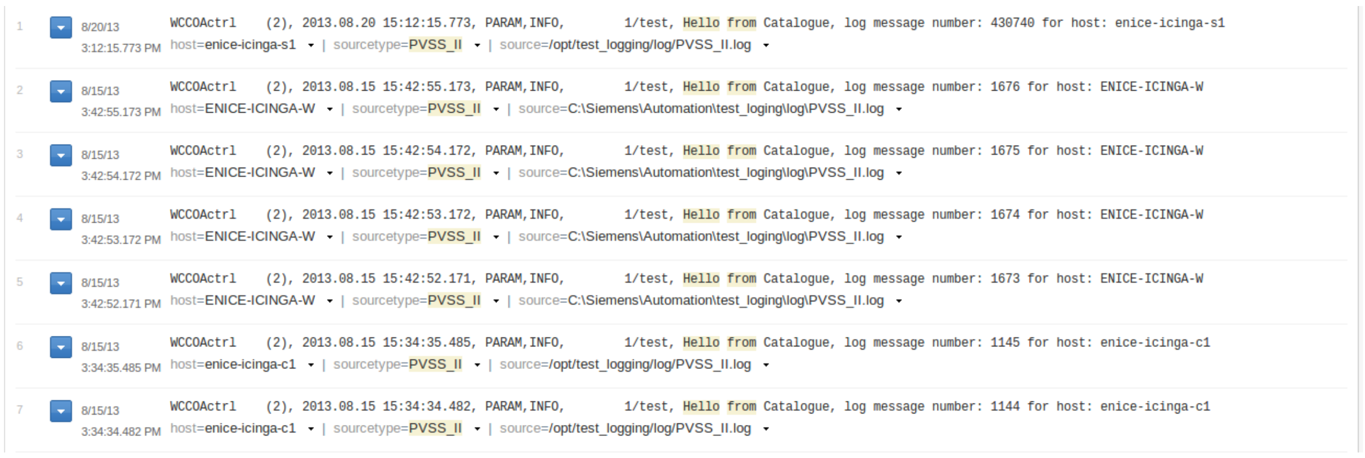
Splunkstorm is a paid tool offering log centralization and analysis. It can be used either with clients only, accessing a web interface hosted by Splunk that shows all gathered data, or with clients and servers, where the servers and web interface are administered by customers. The pricing model is based on amount of data per month. Since this is surely better than not having log centralization and analysis at all, for Splunkstorm we had to evaluate if the features it provided would be of use, in such a way that it justifies the amount of money paid for the service.

3.2 Features and advantages

Ease of setup must be named as one of Splunkstorm strongest suits. If running in client only mode, all that needs to be done is to install the software on a certain machine, and tell it, either via command line or configuration file to get data from a specific log file or folder into the application.

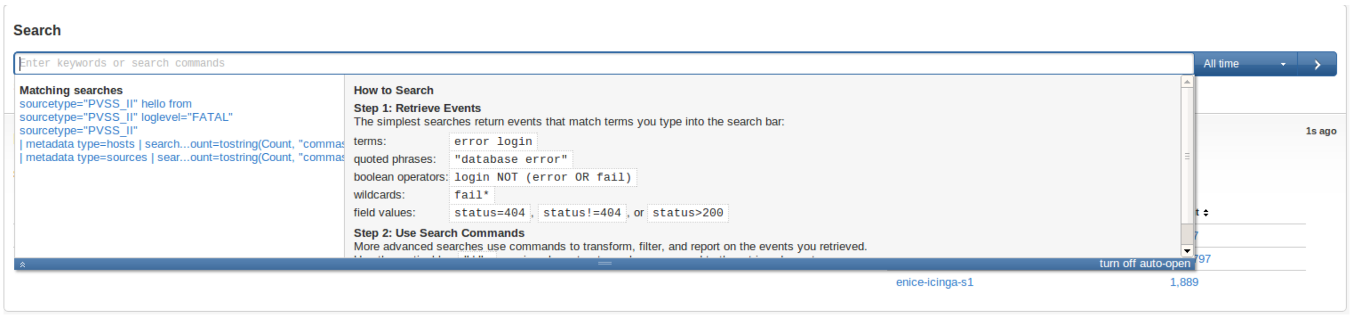
Next come the benefits of log centralization:

Having data all in one place and ordered by time of appearance makes it very easy to visualize and to correlate log events from multiple server, an action that is always needed when debugging an issue.



```
1  8/20/13          WCCOActrl    (2), 2013.08.20 15:12:15.773, PARAM,INFO,       1/test, Hello from Catalogue, log message number: 430740 for host: enice-icinga-s1
   3:12:15.773 PM   host=enice-icinga-s1 | sourcetype=PVSS_II | source=/opt/test_logging/log/PVSS_II.log

2  8/15/13          WCCOActrl    (2), 2013.08.15 15:42:55.173, PARAM,INFO,       1/test, Hello from Catalogue, log message number: 1676 for host: ENICE-ICINGA-W
   3:42:55.173 PM   host=ENICE-ICINGA-W | sourcetype=PVSS_II | source=C:\Siemens\Automation\test_loging\log\PVSS_II.log

3  8/15/13          WCCOActrl    (2), 2013.08.15 15:42:54.172, PARAM,INFO,       1/test, Hello from Catalogue, log message number: 1675 for host: ENICE-ICINGA-W
   3:42:54.172 PM   host=ENICE-ICINGA-W | sourcetype=PVSS_II | source=C:\Siemens\Automation\test_loging\log\PVSS_II.log

4  8/15/13          WCCOActrl    (2), 2013.08.15 15:42:53.172, PARAM,INFO,       1/test, Hello from Catalogue, log message number: 1674 for host: ENICE-ICINGA-W
   3:42:53.172 PM   host=ENICE-ICINGA-W | sourcetype=PVSS_II | source=C:\Siemens\Automation\test_loging\log\PVSS_II.log

5  8/15/13          WCCOActrl    (2), 2013.08.15 15:42:52.171, PARAM,INFO,       1/test, Hello from Catalogue, log message number: 1673 for host: ENICE-ICINGA-W
   3:42:52.171 PM   host=ENICE-ICINGA-W | sourcetype=PVSS_II | source=C:\Siemens\Automation\test_loging\log\PVSS_II.log

6  8/15/13          WCCOActrl    (2), 2013.08.15 15:34:35.485, PARAM,INFO,       1/test, Hello from Catalogue, log message number: 1145 for host: enice-icinga-c1
   3:34:35.485 PM   host=enice-icinga-c1 | sourcetype=PVSS_II | source=/opt/test_logging/log/PVSS_II.log

7  8/15/13          WCCOActrl    (2), 2013.08.15 15:34:34.482, PARAM,INFO,       1/test, Hello from Catalogue, log message number: 1144 for host: enice-icinga-c1
   3:34:34.482 PM   host=enice-icinga-c1 | sourcetype=PVSS_II | source=/opt/test_logging/log/PVSS_II.log
```

Here we can see log events of the same application, PVSS, but from 3 different servers, namely enice-icinga-s1, enice-icinga-w, enice-icinga-c1.



Matching searches
sourcetype="PVSS_II" hello from
sourcetype="PVSS_II" loglevel="FATAL"
sourcetype="PVSS_II"
| metadata type=hosts | search...ount=tostring(Count, "commas
| metadata type=sources | sear...ount=tostring(Count, "commas

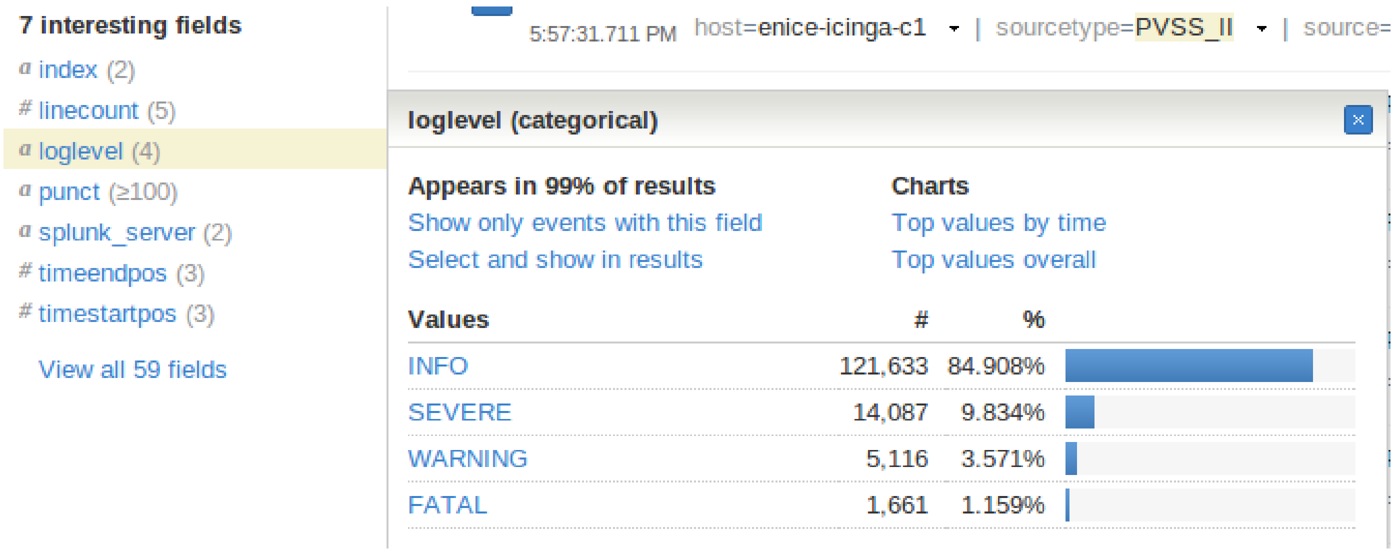
How to Search
Step 1: Retrieve Events
The simplest searches return events that match terms you type into the search bar:

terms:	`error login`
quoted phrases:	`"database error"`
boolean operators:	`login NOT (error OR fail)`
wildcards:	`fail*`
field values:	`status=404`, `status!=404`, or `status>200`

Step 2: Use Search Commands
More advanced searches use commands to transform, filter, and report on the events you retrieved.

The provided search is very powerful, letting you filter log events using complex queries.

Splunkstorm tries to parse log event fields as best as it can. However, custom fields can be added. By providing some examples of values of a certain field the application automatically builds a search expression for it, allowing for review before saving. With this we were able to save the level of severity of each event to a custom field. Having a field saved adds functionality to search for different values of said field and also



Values	#	%
INFO	121,633	84.908%
SEVERE	14,087	9.834%
WARNING	5,116	3.571%
FATAL	1,661	1.159%

increases performance.

3.3 Installation and integration

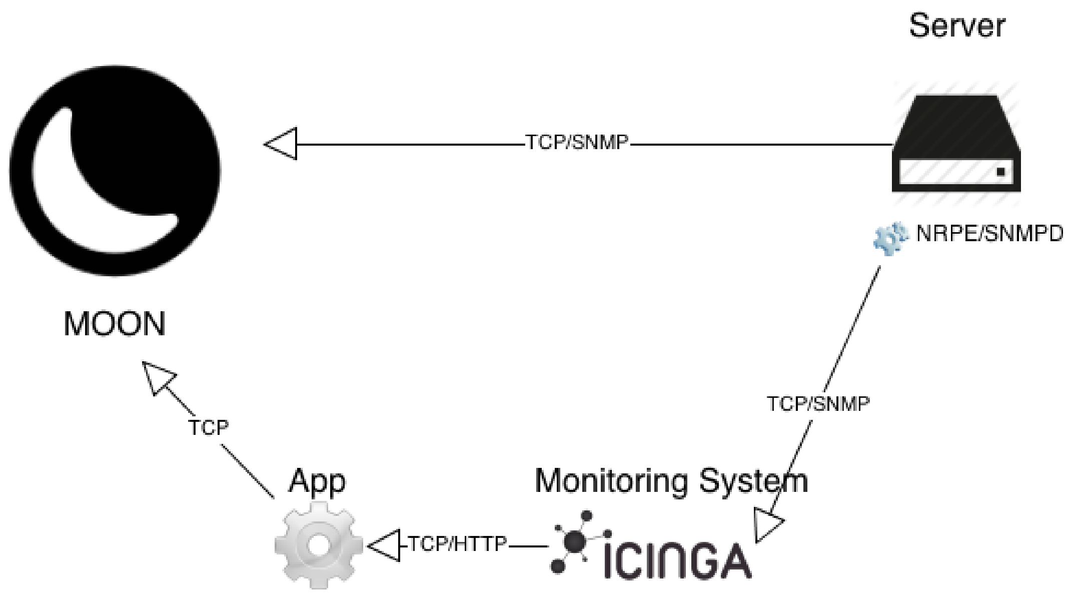
Installation and configuration of splunkstorm is easy, it is supported both in *NIX and Windows systems. Installation and configuration procedure are explained at length here: http://docs.splunk.com/Documentation/Storm/latest/User/AboutforwardingdatatoStorm

4 Conclusions

4.1 Icinga

After understanding all the factors with Icinga it became apparent that the easiest way to replace the FMC agents by querying raw data from the servers using SNMP protocol, which is natively supported in WinCC OA applications.

The longer term solution would be to implement Icinga with a custom program that queries icinga-core using Livestatus. This is still likely to be used, since Icinga provides good features that can extend the current functionality of the present setup.



4.2 Splunkstorm

Splunkstorm is very likely to be implemented. The costs of operating are high, but by collecting only the most important and relevant log files the data volume can be kept reasonable, and as a result keep the price reasonable.

5 Annexes

5.1 A - Installation of Icinga

```
https://wiki.icinga.org/display/howtos/Icinga+and+Oracle+Part1+-
+Installing+Oracle

#### Install oracle support
# yum --enablerepo=slc6-cernonly install oracle-instantclient-basic oracle-
instantclient-devel oracle-instantclient-jdbc oracle-instantclient-odbc oracle-
instantclient-precomp oracle-instantclient-sqlplus oracle-instantclient-
tnsnames.ora cx_Oracle php-oci8 tora

# cd /usr/src/
# wget
http://downloads.sourceforge.net/project/orclib/OCILIB%20Sources/3.12.1/ocilib-
3.12.1-
gnu.tar.gz?r=http%3A%2F%2Fsourceforge.net%2Fprojects%2Forclib%2Ffiles%2FOCILIB%
2520Sources%2F3.12.1%2F&ts=1375103363&use_mirror=garr

# tar xvfz ocilib-3.12.1-gnu.tar.gz

# cd ocilib-3.12.1

#./configure --with-oracle-headers-path=/usr/include/oracle/11.2.0.3.0/client -
-with-oracle-home=/usr/lib64/oracle/11.2.0.3.0/client
# make
# make install

### Install Icinga software

# yum install --enablerepo=rpmforge icinga

Relevant config files:
/etc/icinga/*

##On Linux client, install nagios-nrpe-server:

# yum --enablerepo=rpmforge install nagios-nrpe

Relevant config files:
/etc/nagios/nrpe.cfg

##On Windows client, install nsclient++

http://www.nsclient.org/nscp/wiki/doc/installation

With NRPE support

## Check scripts, on both client and server

# yum install --enablerepo=rpmforge nagios-plugins nagios-plugins-ping nagios-
plugins-ssh nagios-plugins-disk nagios-plugins-load nagios-plugins-swap nagios-
plugins-procs nagios-plugins-http nagios-plugins-dummy nagios-plugins-nrpe

#### Idoutils - used for storing info in a db:
Idomod: Icinga event manager, writes in a unix socket in a language ido2db
understands
Ido2db: Reads from idomod's socket and writes to a specified database(mysql,
psql, oracle supported)
```

```
idoutils from rpmforge repo didn't have oracle compatibility, I had to download
the same all of the icinga package with the same version(1.8.4) and compile
idoutils from that

# cd /usr/src/
# wget http://downloads.sourceforge.net/project/icinga/icinga/1.8.4/icinga-
1.8.4.tar.gz?r=http%3A%2F%2Fsourceforge.net%2Fprojects%2Ficinga%2Ffiles%2Ficing
a%2F1.8.4%2F&ts=1375103171&use_mirror=switch
# tar xvfz icinga-1.8.4.tar.gz
# yum install icinga-idoutils-libdbi-mysql.x86_64 --enablerepo=rpmforge
# ./configure --with-command-group=icinga-cmd --enable-idoutils --enable-oracle
# make idoutils
# make install-idoutils
# /etc/init.d/icinga start
# /etc/init.d/ido2db start

Relevant config files:
/etc/icinga/idomod.cfg
/etc/icinga/ido2db.cfg
vim /etc/icinga/modules/idoutils.cfg
/etc/init.d/icinga
/etc/init.d/ido2db

### Install livestatus
http://docs.icinga.org/latest/en/int-mklivestatus.html

# yum --enablerepo=rpmforge install check-mk-livestatus
# /etc/init.d/icinga/restart
# chmod 666 /var/spool/icinga/live.sock

Relevant config files:
/etc/icinga/modules/livestatus.cfg


### Install xinetd
# yum install xinetd
# /etc/init.d/xinetd start

Relevant config files:
/etc/xinetd.d/livestatus
```

5.2 B – Xinetd livestatus configuration.

```
service livestatus
{
       type          = UNLISTED
       port          = 6557
       socket_type   = stream
       protocol      = tcp
       wait          = no
# limit to 100 connections per second. Disable 3 secs if above.
       cps              = 100 3
# set the number of maximum allowed parallel instances of unixcat.
# Please make sure that this values is at least as high as
# the number of threads defined with num_client_threads in
# etc/mk-livestatus/nagios.cfg
        instances       = 500
# limit the maximum number of simultaneous connections from
# one source IP address
        per_source      = 250
# Disable TCP delay, makes connection more responsive
       flags            = NODELAY
       user          = nagios
       server        = /usr/bin/unixcat
       server_args      = /var/lib/nagios/rw/live
# configure the IP address(es) of your Nagios server here:
#      only_from        = 127.0.0.1 10.0.20.1 10.0.20.2
       disable              = no
}
```

5.3 C – sockey.py

```
#!/usr/bin/python
#
# Sample program for accessing the Livestatus Module
# from a python program
import socket

socket_path = "/var/spool/icinga/live.sock"

# Create a UNIX socket object.
s = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)

# Conect to livestatus's socket.
s.connect(socket_path)

# Write command to socket.
s.send(
    "GET services\n"
    "Columns: host_alias display_name state plugin_output\n"
    "Filter: host_alias != localhost\n"
    "OutputFormat: json\n"
    )

# Important: Close sending direction. That way
# the other side knows we are finished.
s.shutdown(socket.SHUT_WR)

# Now read the answer
answer = s.recv(100000000) # this is max buffer size
print answer
```

5.4 D – tcp.py

```
#!/usr/bin/python
#
# Sample program for accessing the Livestatus Module
# from a python program
from collections import defaultdict
import simplejson as json
import socket
from prettyprint import pp

host = 'enice-icinga-s'
port = 6557

# Create a TCP socket object.
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Conect to livestatus's socket.
s.connect((host, port))

# Write command to socket.
s.send(
    "GET services\n"
    "Columns: host_alias display_name last_check plugin_output state perf_data
process_performance_data\n"
    "Filter: host_alias != localhost\n"
    "OutputFormat: json\n"
    )

# Important: Close sending direction. That way
# the other side knows we are finished.
s.shutdown(socket.SHUT_WR)

# Now read the answer.
answer = ''
answer_chunk = True
while answer_chunk:
    answer_chunk = s.recv(4096)
    answer += answer_chunk

# Load the data in a dict(hash)
answer_json = json.loads(answer)
answer_ddict = defaultdict(lambda : defaultdict(dict))

# Add some meaning and structure to the displayed values.
for row in answer_json:
    answer_ddict[row[0]][row[1]]['last_check'] = row[2]
    answer_ddict[row[0]][row[1]]['plugin_output'] = row[3]
    answer_ddict[row[0]][row[1]]['state'] = row[4]
    answer_ddict[row[0]][row[1]]['perf_data'] = row[5]
    answer_ddict[row[0]][row[1]]['process_performance_data'] = row[6]

# Print it nicely.
pp(answer_ddict)
```

6 Bibliography

Icinga: http://docs.icinga.org/latest/en/

Event handlers: http://docs.icinga.org/latest/en/eventhandlers.html

Performance data: http://docs.icinga.org/latest/en/perfdata.html

Idoutils: http://docs.icinga.org/latest/en/db_components.html

Livestatus: http://mathias-kettner.de/checkmk_livestatus.html

Oracle tablespace: https://wiki.icinga.org/display/howtos/Icinga+and+Oracle+Part3+-+Configuration#IcingaandOraclePart3-Configuration-createtablespacesanduser

Splunkstorm: http://docs.splunk.com/Documentation