Thesis

Distribution and configuration of agents for NMS in a reasonable time

Authors: Robin Jonsson & Simon Blixt
Supervisor: Peter Adiels
Semester: Autumn 2013
Course code: 2DV00E

Linnæus University
Sweden

Abstract

With this paper the authors intended to simplify deployment and management of monitoring agents for a Network Monitoring System. The authors found interest on the subject since the time consumed to deploy and manage agents was found to be very inefficient. During a lecture with the Swedish based company Op5 AB at the Linnaeus University in Kalmar, Sweden, the authors presented the complex of problem. The lecturer showed great interest in a solution on the subject and the authors of this paper found it to be a great thesis subject for the Bachelor degree in Computer Science.

By the year of 2016 it is expected that the number of network connected devices will grow threefold, there will be four times as much IP traffic and the data storage demand will increase tenfold. This growing demand will also affect the requirement on the Network Monitoring System and in turn the monitoring agents.

In this paper the authors created a baseline, which consisted of a timing regarding the time consumption for manual deployment, configuration and management of the monitoring agents. The authors also developed an automated way for deployment, configuration and management of monitoring agents by integrating a Content Management Software called Puppet, combined with several scripts. To simplify the management and deployment furthermore a widget was developed for Op5’s Web based User Interface called Ninja.

The developed solution was measured against the baseline and a result regarding time consumption was presented. The result fell into a discussion on the subject of automatization and the time savings that it may result in due to less frequent human errors and a less repetitive work processes.

The presented solution is tested and developed for Centos. But it should also work for other Linux distributions with varying modifications, which is briefly discussed in this paper. The solution and code presented in this paper may be updated and is available at Github[1].

Keywords: Network Monitoring System, Op5, Nagios, NMS, Content Management System, CMS, Puppet, System integration, Automatization, Deployment of agents, Management of agents

[1] https://github.com/jesajah/RoSi

Foreword

We would like to thank Marcus Wilhelmsson, Lecturer in Computer Science at Linnaeus University, for the contribution of hardware and consulting during this paper. The authors would also like to thank Peter Andersson, Product Manager at Op5, for his guidance and faith.

Linnæus University
Sweden

Contents

1. Introduction ___ ***1***
1.1 Background ___ 1
1.2 Purpose ___ 1
1.3 Previous research ___ 2
1.4 Target audience ___ 2
1.5 Problem ___ 2

2. Theory ___ ***3***
2.1 Network Monitoring System ___ 3
2.2 Configuration Management System ___ 4
2.3 Other theory ___ 6

3. Method ___ ***7***
3.1 Scientific approach ___ 7
3.3 Implementation ___ 7
3.4 Reliability ___ 11

4. Result ___ ***13***
4.1 WEB GUI ___ 13
4.2 Puppet ___ 15
4.3 Scripts ___ 17
4.4 Test results ___ 18
4.5 Overview ___ 20

5. Discussion ___ ***25***
5.1 WEB GUI ___ 25
5.2 Puppet ___ 26
5.3 Scripts ___ 27
5.4 Test results ___ 27
5.5 Other discussions ___ 28

6. Conclusion ___ ***30***
6.1 Further research ___ 31

References ___ ***32***

Attachments _____ *36*
A – WEB GUI _____ i
B - Manifest for puppet_script _____ v
C – Manifest for NRPE _____ vi
D - Manifest for monitor_script _____ ix
E – Add check on nodes _____ x
F - Add hosts _____ xiii
G - Add services linux _____ xiv
H - Add nodes puppet _____ xvi
I - Remove nodes puppet _____ xvii
J - Puppet code for runinterval and usecacheonfailure _____ xix

1. Introduction

Deployment and maintenance of software may be a very time consuming and complex duty. As more and more systems are put into production and the IT infrastructure is growing, the more it is required that repeatable tasks are kept to a minimum. [1] By the year of 2016 it is expected that the number of network connected devices will grow threefold, there will be four times as much IP traffic and the data storage demand will increase tenfold. [8]

According to the authors of this paper the increasing demand indicates growing requirements on the Network Monitoring System as more and more systems interconnect. The growing requirements to distribute the agents for the Network Monitoring System might lead to increased administration for the IT personnel.

The authors of this paper will focus on how to distribute the agents of a Network Monitoring System and how to make changes to the agents using a Content Management System combined with several scripts. When a solution has been created and evaluated, the authors are going to try to find a way to integrate the solution with a web based graphical user interface. The integration is created to interconnect with the Network Monitoring System, and thereby simplify the deployment and management of agents.

1.1 Background

In the beginning of 2013 the Swedish based company Op5 AB held a lecture at the Linnaeus University in Kalmar, Sweden. During this lecture a discussion occurred on the subject of a simple way to distribute and manage Network Monitoring System agents on the nodes. This was a function the authors had been longing for during earlier courses at the University, since the time consumed to configure multiple agents felt very inefficient. The lecturer from Op5 showed great interest in a solution on the subject and by the end of the lecture the authors of this paper and the lecturer from Op5 exchanged contact information to be able to keep in touch.

1.2 Purpose

The authors' intention and expectation is to produce a solution on how it is possible to manage and deploy Network Monitoring System agents and services on monitored nodes in a network. When a solution is created, the authors intend to create a graphical user interface for the solution. This can be integrated to the Network Monitoring System to make the solution more user-friendly. By succeeding with this the authors hope to reduce the time and ease the management of Network Monitoring System agents for administrators in a large IT environment.

1.3 Previous research

The authors have not found any published scientific research on the specific subject. The most related work to this paper is a published article where the author has published a roadmap to integrate Puppet with Nagios [22]. There is also scientific research regarding Configuration Software Systems such as Puppet and Network Monitoring Systems like Nagios. The authors of [19] use the Content Management System, Puppet to accomplish cluster deployment and recovery to simplify cluster management. In [20] the authors use Nagios as a ground to develop a more user-friendly and efficient Network Monitoring System. The authors of the paper published some approaches on how to accomplish it.

1.4 Target audience

This paper is addressed primarily for administrators and personnel working with Network Monitoring Systems like Nagios in large IT infrastructures. The paper could also be of interest for a reader involved or with an interest in system integration, deployment of software and how to simplify the distribution in a time efficient manner. Basic Unix / Linux server administration is recommended.

1.5 Problem

"How is it possible to simplify deployment of agents and management of services on monitored nodes in a mixed IT infrastructure in a reasonable time?"

The authors define reasonable time as in "How much more time would it take to manage and deploy the agents manually?". Mixed IT infrastructure is defined as a heterogeneous IT infrastructure with different hardware and Linux operating systems. The opposite would be a homogenous IT infrastructure, where everything is identical.

2. Theory

The theory section concerns to give the reader basic knowledge to meet the subsequent sections of this paper. This section treats Network Monitoring, Content Management System and other related theory.

2.1 Network Monitoring System

2.1.1 Overview

Network Monitoring System, also known as NMS, is used to monitor an IT environment in order to get information regarding network performance, security situation, hardware and software status presented within a graphical web interface. [2, 3] It is also possible to write own plugins for IT environment specific needs. [2] Todays large IT infrastructures have software and hardware from a lot of different vendors. By monitoring the hardware and software of the IT infrastructure it increases the chance of detecting, for example, a potential disk crash before it affects the IT infrastructure. [3]

2.1.2 Agents

An agent is used on monitored nodes to allow execution of locally available resources like CPU load, memory and disk usage. The collected data is pushed back to the Network Monitoring System.[4] There are a few available agents, both for Windows and Linux. One example for Windows systems is Nsclient++ and for Linux/Unix systems the most widely used is NRPE (Nagios Remote Protocol Executor) [6, 7]. Op5 provides an own version of NRPE to suit an environment with Op5 monitor. [36]

2.1.3 Op5

Op5 is an Open Source Network Monitoring System created in 2003, and established as the company Op5 AB in early 2004. It is based on Nagios, initially called NetSaint. Nagios is a world-known Open source monitoring system launched in 1999. Nagios provides monitoring, alerting, response of alerts, reporting, scheduling maintenance, planning and many more functionalities. [5, 6]

Op5 uses the Nagios core combined with three important own-created solutions: Ninja, Merlin and Nacoma [7, 28]. Ninja is an acronym for *Nagios Is Now Just Awesome* and poses Op5’s web interface with the aim to be the most powerful and useful open source web front end for Nagios [34]. Merlin is an acronym for *Module for Effortless Redundancy and Load balancing in Nagios* and provides Op5 with the database, which works as a backend for Ninja [35]. Op5 also provides Nacoma, which is a *Nagios Configuration Manager*. Nacoma is an Open Source configuration tool for Op5 Monitor, written in PHP and uses MYSQL as backend. Nacoma provides an easy way to manage configuration files and propagation of hosts via an API. [28]

2.2 Configuration Management System

2.2.1 Overview

A Configuration Management System (CMS) provides a solution for automating configuration tasks on computer systems. A Configuration Management System may for example be useful if you would like to create and configure three identical servers. The packets to be installed are specified on the Configuration Management System and distributed to the nodes. It may also be used to distribute script for execution on multiple nodes. [9]

2.2.2 Puppet

Puppet is a Configuration Management System used to automate distribution of resource configurations across an IT infrastructure. Puppet makes it easy to automate functions which lead to simplified provision, configuration, and management of infrastructures throughout its lifecycle. [10] Puppet can run either in a client-server or stand-alone mode. It has support for managing Unix, OSX, Linux and Windows platforms. Puppet consists of three components [15]:

- Deployment
- Configuration Language and Resource Abstraction Layer (RAL)
- Transaction Layer

Deployment - Puppet is mostly used in a client-server mode, where the server with the Puppet software is called "Puppet master" and the nodes which are to be managed are called "Puppet nodes". The communication and connection between the Puppet master and the nodes are made via an encrypted and authenticated connection using standard SSL. [15]

Configuration Language - The configuration language Puppet uses is a declarative language. The declarative language is used to define configuration items, also called "resources". A declarative language means that Puppet makes statements about the state of the configuration, for example: it declares that the package NRPE should be installed, and the service NRPE should be started. In this way the administrators who use Puppet only need to declare how the nodes should be configured regarding packages and services. Puppet's work is thereby to make these states to be achieved. This is done by abstracting the hosts' configuration into resources. [15]

If a system administrator does not use a CMS like Puppet he needs to make a lot of steps for just a simple thing like installing NRPE on five linux nodes, each with a different operating system. The system administrator first needs to connect to the required host, make a check to see if NRPE is installed. If it is not, the system administrator needs to use the appropriate command for the current platform to install NRPE. As a last thing to do the system administrator checks if it all went well. These steps will be easier and less time consuming by using Puppet. The only thing the system administrator needs to configure, besides installing the puppet agents on the nodes and some minor puppet configurations, is the code presented in *Code 2.2.2.1*. [15]

```
package { "NRPE":
ensure => present,
}
```

Code 2.2.2.1 - Code to make sure the package NRPE is installed

A Puppet resource begins with a type (packages, services, cron jobs, mount etc.), which declares the sort of resource that is being managed. Afterwards a series of attributes are specified which for example makes it possible to check if a service is running or not. An example of a resource construction is shown in *Code 2.2.2.2*. [15]

```
type { title:
attribute => value,
}
```

Code 2.2.2.2 - A Puppet resource construction

To use Puppet as it should be used modules and classes are the right way to go. Modules are reusable code and data which can easily be loaded into files like the manifest file site.pp. In the modules classes should be used to structure the code. The flow control is a lot easier if classes are used. Classes are a collection of resources like *Code 2.2.2.1* which Puppet can apply as an unit. [29]

RAL - When a resource is created by the system administrator, Puppet takes care of the rest when the nodes connect. By knowing how different platforms and operating systems manage certain types of resources, Puppet makes the configuration, administration and installation. This is possible by different providers of the "type". If the type is specified as package there are more than 20 providers like yum, aptitude, pkgadd, port and emerge. To decide which provider to use Puppet uses Facter. Facter is a tool that returns information about the node, for example what Operating System it is running. On the basis of the information from Facter, Puppet chooses the correct package provider, for example aptitude for Ubuntu and other Debian based distributions or yum for Redhat Enterprise Linux based distributions like Centos and Fedora. If the package is already installed, Puppet will not do anything. If it is not installed Puppet will install the package. [15]

Transactional Layer - Puppet describes the Transactional Layer as the engine.

It covers the process of configuring each host, operations like interpret and compile the system administrations configuration, send the compiled configuration to the agents, apply the configuration on the agents and report the results to the Puppet master. [15]

2.3 Other theory

2.3.1 Open source

Open Source is referring to “Free” software. Free should be read as free as in freedom, not “for free”. Programs that are licensed under an Open Source license are free to use, modify and redistribute. [17]

2.3.2 Vmware Esxi

Vmware Esxi is a bare-metal hypervisor which makes it possible to run multiple virtual machines on a single set of hardware. Esxi is built to require minimal configuration and to be up and running in just a couple of minutes. [14]

2.3.4 Network File System

Network File System (NFS) allows remote hosts to mount file systems over the network and interact with those systems as though they are mounted locally. This enables system administrators to consolidate resources on centralized servers on the network. [18]

2.3.5 Graphical User Interface

A Graphical User Interface (GUI) displays graphical components for a user. The graphical components depend on the user’s requested information. The user can mostly interpret with the GUI and for example add, change and delete objects. [25]

3. Method

The method section presents the scientific approach, implementation and reliability. The scientific approach presents the methods to be used for the implementation. The implementation section presents how the solution is being developed and how it is implemented in the test environment. The method section also explains details that may affect the result and its reliability.

3.1 Scientific approach

The authors of this paper used a mixed method. By using a qualitative approach data is derived from empiricism and accordingly is able to draw conclusions on how it is possible to simplify management and deployment of agents and services on monitored nodes in a mixed IT infrastructure. By using a quantitative approach the authors measured the outcome of the produced solution. [16]

The authors intended to use an inductive and a deductive approach. The inductive approach refers to create theory from observations and results. The observations that have been made showed that management and deployment of agents are a time consuming chore. By using this theory, the authors pertained to use a deductive approach to bring forward a method to simplify the administration of agents in Nagios based Network Monitoring Systems. The deductive approach aims to draw conclusions from observations regarding the time consumption. [37]

3.3 Implementation

3.3.1 Lab environment

To be able to perform the test, a lab environment was set up, presented as a topology in *Figure 3.3.1.1*.

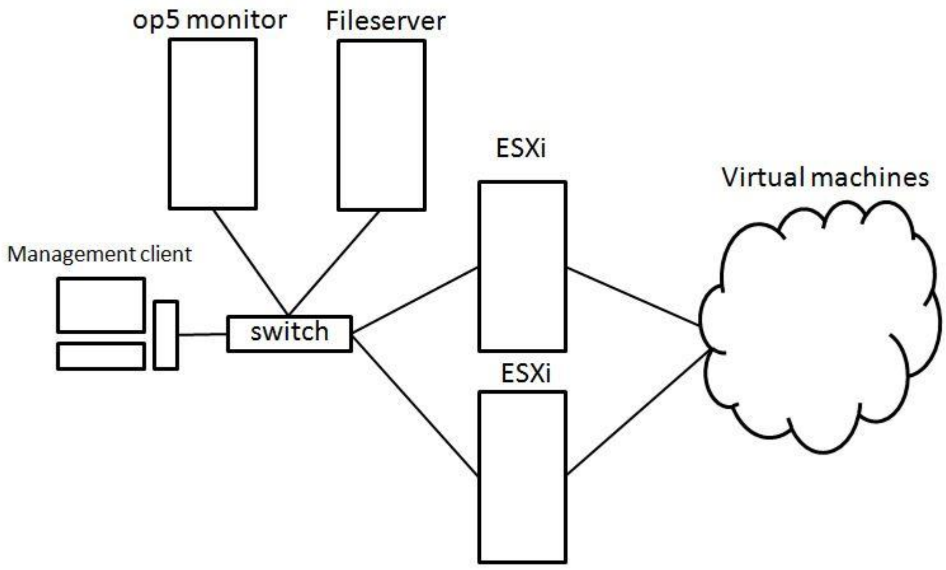


Figure 3.3.1.1 - Topology of the lab environment

The monitored nodes were run virtually since it would otherwise be required to have access to a much larger amount of physical hardware. The virtual machines were run on two unclustered Vmware Esxi 5.1 machines with 9GB of RAM each. The virtual machines are presented in *Figure 3.3.1.2*.

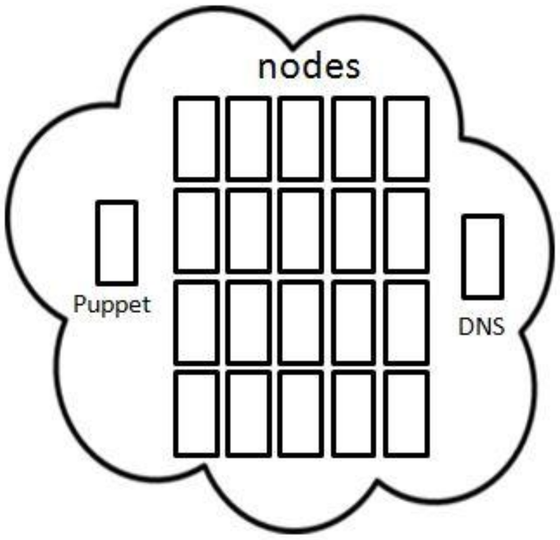


Figure 3.3.1.2 - Virtual machines

The virtual nodes consisted of a total of 20 machines. These machines ran Centos 6.4. The amount of RAM for each node was limited to 512 MB. One of the Esxi machines also hosted a DNS server running Ubuntu 12.10 with Bind9 and another machine running Centos 6.4 hosting the Puppet master. These two machines had 1GB of RAM each. The DNS server was configured with DNS records for all the nodes and other devices in the test environment.

As mentioned in the introduction section of this paper, the work was done in collaboration with Op5 AB. Accordingly, the choice of the NMS was their monitoring software called Op5 monitor. By the time of writing the latest version was 6.07 which is built on Nagios Core 4. [11] The Op5 monitor server was run on a physical machine with 2GB of RAM. There was a physical file server with 2GB of RAM. The file server contained the files to be distributed to all nodes with Puppet configured. The file server also contained one file which the nodes wrote their hostnames to when they were done and the directory /archived_services_and_hosts/ with the purpose to store and archive previous runs. The file server used Centos 6.4 and exported the files via NFS.

3.3.2 Flow chart

The flow chart, presented in *Figure 3.3.2.1*, served as a model of how the whole solution was meant to work. It began with an user input via the GUI of the NMS. The input contained information regarding which nodes should be configured with agents and which services should be added and configured on the nodes. The input was also used by the NMS to add the nodes to the GUI and to add the services to those nodes.

The information regarding nodes and services were stored in two files which the Puppet had a subscription to. Namely if the file existed Puppet should execute the modules on the Puppet nodes. One text file contained the hostnames for the nodes, and the other text file contained the services to be added or changed. If a new service was added, the script for the service should be uploaded too. These three files; hostnames, services and the script, were uploaded by the GUI to a file server located in the environment.

Puppet had the role to contribute with modules which contained tasks (classes), such as to ensure the file server was mounted, scripts were executed, automated the distribution or configured the agents. The NMS had a module on the Puppet master for the tasks it was expected to do. *Figure 3.3.2.1* presents the flow chart of the solution in four easy steps. A short explanation of the steps may be seen to the left of the figure.

1. User input - indicates nodes and services to be changed or added.

2. Modules execution - Execute scripts etc. on the NMS and nodes.

3. The nodes sends monitoring-data to the NMS.

4. The NMS presents the data on the GUI.

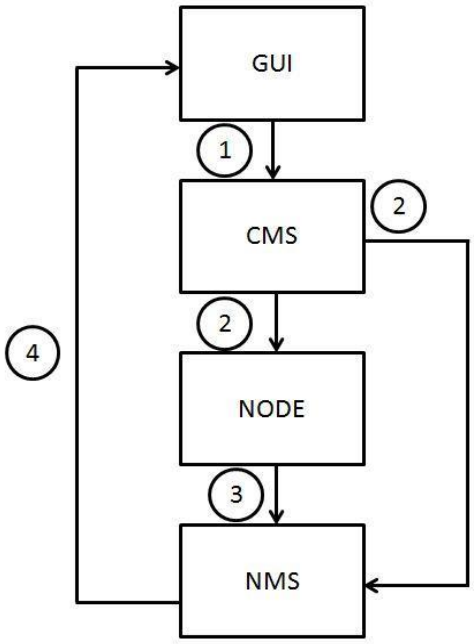


Figure 3.3.2.1 – Flow chart

3.3.3 Test suite

The test suite was composed by a total of three tests. One baseline test and two tests in order to stress test the solution presented by the authors. The solution was measured against the baseline test regarding time consumption. The baseline test had a major reliance on the person performing it, but was meant to work as a comparison to the authors' presented solution. The baseline test was performed two times by the authors and an average value is presented. Major differences are discussed and analyzed. The steps are presented in *Table 3.3.3.1*.

#	**Test description**
1.	Install NMS Agent on 10 nodes.
2.	Add hosts to the NMS via the GUI with services included.
3.	Add a specified new service for 10 nodes.
4.	Add one new service to the NMS GUI.
5.	Change a specified service for 10 nodes.

Table 3.3.3.1 - Baseline test suite

The automated solution tests is presented in *Table 3.3.3.2* and consisted of the same steps as presented in *Table 3.3.3.1*, but some of the steps were merged due to the functionality of the automated solution. Each test was performed three times and an average value is presented. Major differences are discussed and analyzed. To be able to see the difference an additional test was performed with 20 nodes. By performing the third and last test the authors strived to show the amount of nodes do not result in a linear increase in time consumption and may thereby demonstrate the scalability of the solution.

#	**Test description**
1.	Install NMS Agent on 10 or 20 nodes and add hosts to the NMS including with specified services.
2.	Add a specified new service for 10 or 20 nodes and add it to the NMS GUI.
3.	Change a specified service for 10 or 20 nodes.

Table 3.3.3.2 - Automated test suite

3.4 Reliability

As mentioned in section *3.3.1 Test environment*, the monitored nodes and the Puppet master were running as virtual machines. This fact should not affect the outcome of the tests. Since Bash is available on most Linux and Unix based operating systems the authors chose this as the only script language for the Linux nodes [21]. The Linux distribution of choice on the nodes was as mentioned in *3.3.1 Test environment*, namely Centos 6.4. This meant that the default package manager was yum which handles rpm packages [24]. This was specified in the Puppet configuration. Which means if the reader intends to use a non “rpm based” distribution, some modification will be required. Due to time constraints this was not covered in this paper. It is possible to use variables in Puppet, and thereby simplify and make the Puppet configuration more general [30]. But due to time constraints the authors have chosen not to implement it.

In comparison between the three Configuration Management Systems Chef, CFengine and Puppet, [12] found out that Puppet has by far the largest user community. According to [12] Puppet also has a very powerful language with the ability to make configurations with little effort. It also stated that Puppet’s documentation has a good structure. Based on [12] and the fact that Puppet is an Open Source software that has support for all the major operating systems and architectures [13], the authors of this paper have chosen Puppet as the CMS.

The authors are aware that if an administrator were managing a large IT environment, a manual method would probably not be considered. The administrator would most likely go with writing an own script or to use an existing solution to propagate the changes to the nodes. To manage all this through one straightforward user interface as in the presented solution in this paper could still be considered convenient.

The timing during the tests was done manually using a stopwatch. The timing began as the activities specified in section *3.3.3 Test suite* begun. The timing was stopped when all of the nodes had got the specified agent, service or changed parameters and after the results had been verified in the NMS. The purpose of the timing was to see differences between the baseline and the solution presented in this paper. The timing was also used to present how the solution's scalability is, as presented in section *3.3.3 Test suite*.

In section *1.5 Problem* the authors presented the problem of this paper and claimed a mixed IT infrastructure would be used. But since all of the nodes were executed in virtual machines, the IT infrastructure could not be considered to be "mixed" as far as hardware is concerned. The authors do not believe the results would be any different with different hardware. The solution is most likely not being completely compatible with other Linux distributions. But the authors of this paper do not see that as a big problem because the configuration to change should not be complicated. When comparing Linux distributions some of the main differences for this paper's solution are the package provider, Selinux availability and NRPE for Op5 directory path.

4. Result

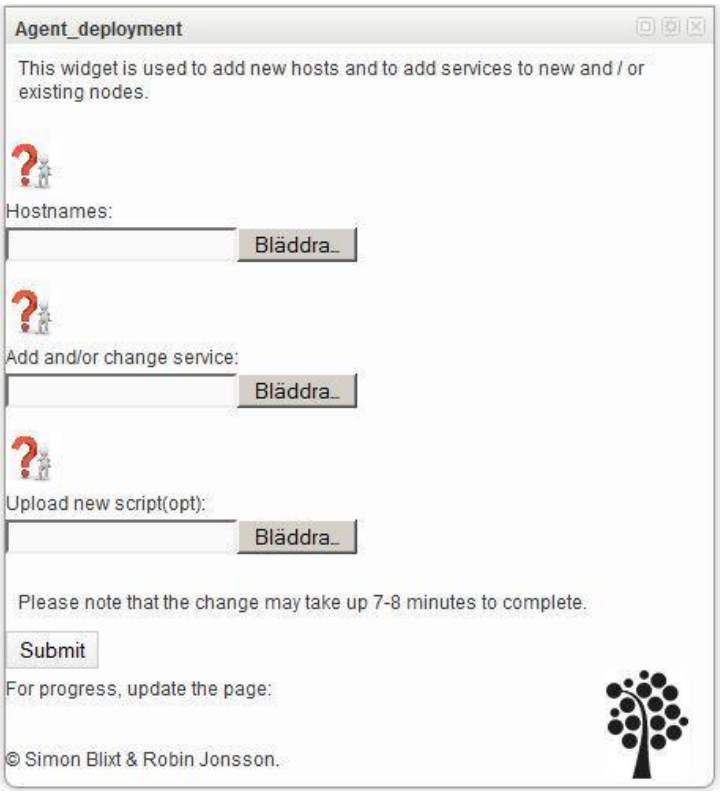
In this section the results that have come to light of the analyses performed in the earlier chapter is presented.

4.1 WEB GUI

The authors have created a widget, presented in *Figure 4.1.1*, for Op5's web based Graphical User Interface called Ninja. The widget is written in HTML and PHP[1] and strives to give the user a more user friendly experience while deploying agents and managing services. From the user's point of view, the widget shows three form boxes which ask the user to upload files to the shared storage. The upload location and purpose is specified in *Table 4.1.1*. The "Hostnames" field prompts the user to upload a file specifying which nodes the chosen services affect. "Add and / or change service" prompt the user to upload a file specifying which services to add or change on the nodes from the previous upload function.

The last field, "Upload new script (opt)" prompts the user to upload a new script. This field gives the user an opportunity to add a new script to all of the specified nodes. The script's service-command and arguments should of course then be specified in the file containing the services. After the user has pressed submit the PHP script echoes if the file upload was accepted or denied depending on the file type and file size. It also echoes the destination of the files and furthermore the progress. The question mark on top of the fields gives the user an example configuration of what information is expected in each field.

[1] Attachment *A - WEB GUI*



Agent_deployment

This widget is used to add new hosts and to add services to new and / or existing nodes.

Hostnames:

Add and/or change service:

Upload new script(opt):

Please note that the change may take up 7-8 minutes to complete.

For progress, update the page:

© Simon Blixt & Robin Jonsson.

Figure 4.1.1 - Widget for op5

Form name	**Format**	**Location**
Hostnames	linuxnode1.example.com randomcomputer2.example.com Should be specified in .txt no larger than 20kB.	/mnt/upload/hosts_op5.txt
Add and / or change service	command[check_file_size]=/opt/plugins/check_file_size.sh --maxwarn 1000000000 --maxcrit 2000000000 /tmp Should be specified in .txt and no larger than 20kB.	/mnt/upload/checks.txt
Upload new script (opt)	check_example.sh Should be specified in script file and no larger than 20kB.	/mnt/scripts/check_example.sh

Table 4.1.1 - Web GUI

4.2 Puppet

Puppet has been configured with three modules. *Table 4.2.1* presents the different modules and a short description. One module is called "puppet_script". The module executes a script, which will add the specified nodes from the uploaded file to the Puppet configuration file. The nodes will there have module "NRPE" included. The task for this module is to ensure NRPE is installed on the nodes specified in the uploaded file and the services specified in checks.txt is added or changed. The module "puppet_script" also triggers a script to clean and remove temporary changes and files created during the run.

The module "NRPE" ensures that the file server is mounted properly. The module will install any missing dependencies for NRPE, install NRPE and apply the NMS as an "allowed_hosts" under the NRPE configuration file. The service NRPE will be restarted when it is done. The module also has the mission to execute a script to add and change services locally to the specified nodes. The services information are gathered from the input specified in the file uploaded from the WEB GUI.

The module for the NMS is called "monitor_script" and will trigger a script containing bash code that uses Nacoma, to add hosts and corresponding services.

Name	**Files**	**Nodes**	**Description**	**Attachment**
puppet_script	manifests/init.pp files/add_nodes_puppet.sh files/remove_nodes_puppet.sh	Puppet	If the file hosts_op5.txt exists the module executes the script add_nodes_puppet.sh which adds nodes to the puppet configuration file (site.pp). It also executes the script remove_puppet.sh when the file /mnt/done.txt[1] is changed.	*B - manifest for puppet_script*
NRPE	manifests/init.pp files/add_check_on_nodes.sh	Linux	Ensures that the fileserver is mounted. Installs NRPE, and its dependencies, from a package accessible from the file server. Allows communication from NMS to the node via NRPE and then restarts the service NRPE. Run script to add new, or change existing, services in NRPE if the file /mnt/upload/checks.txt[1] exists.	*C - manifest for NRPE*
monitor_script	manifests/init.pp files/add_hosts.sh files/add_services_linux.sh	NMS	Run script to add new hosts to the NMS if the file hosts_op5.txt exist, and add the specified services in checks.txt to these hosts, if checks.txt[1] exists.	*D - manifest for monitor_script*

Table 4.2.1 - Overview of Puppet modules

[1] Note: The execution of scripts in a module is not done right away, it is executed on the next puppet run.

4.3 Scripts

Several scripts have been written to contribute with tasks which Puppet is not capable of producing, or in cases where a script is more suitable. *Figure 4.3.1* presents all scripts and relationships with different files.

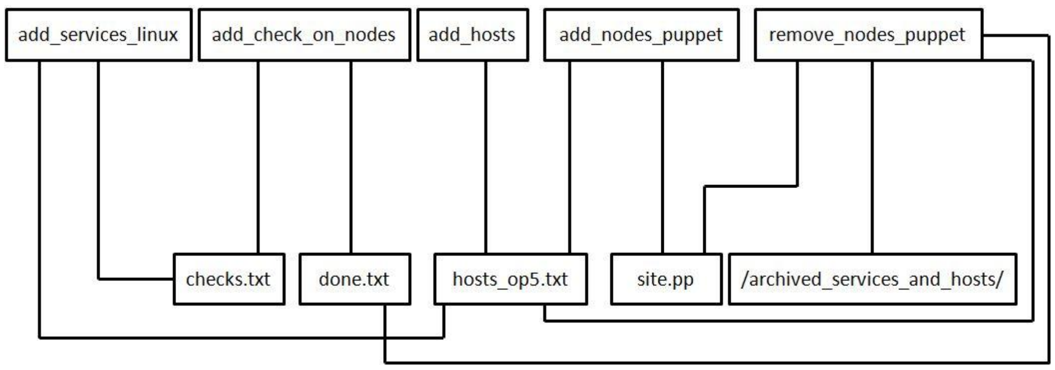


Figure 4.3.1- Script overview

4.3.1 Add check on nodes (add_check_on_nodes.sh)[1]

The nodes add the services to the NRPE configuration by taking the services specified in the uploaded file checks.txt, presented in *Table 4.1.1 - Web GUI.* The script checks those services against the already existing ones, in Op5's NRPE version located in /etc/nrpe.d/op5_commands.cfg. If a service already is present but with new arguments the service is replaced with the new arguments specified in checks.txt.

The script also checks if a file was uploaded to /mnt/scripts/. If it is a file uploaded to the location, the script extracts the filename of it and copies it to /opt/plugins/<filename.fileextension>. The script also makes it executable. If new services have been added or old ones changed, the script restarts the NRPE service. As a last thing to do, the script checks whether all services have been added or not. If all services are present, the hostname of the node is echoed to the file /mnt/done.txt. Which is later used by the script, remove_nodes_puppet.sh.

4.3.2 Add hosts (add_hosts.sh)[2]

Add_hosts.sh simply adds hosts to the NMS from the file hosts_op5.txt which the WEB GUI has uploaded to the file server. The script checks if the hosts already exist or not. The existing hosts are read from /opt/monitor/etc/hosts.cfg and are saved in a variable, which is compared to the hosts specified in hosts_op5.txt. If the new hosts are not existing ones,

[1] Attachment E – *Add check on nodes*
[2] Attachment F – *Add hosts*

they will be added via Nacoma. If new hosts have been added the save command will be executed for the Op5 monitor.

4.3.3 Add services linux (add_services_linux.sh)[1]

The script reads the file hosts_op5.txt and checks.txt. For each node, the script adds the services specified in checks.txt to the node in the WEB GUI, if they are not already present. The script also adds the two standard services check_ping and check_ssh, if they do not exist. At last everything is saved if changes have occurred.

4.3.4 Add nodes puppet (add_nodes_puppet.sh)[2]

Add nodes from the file hosts_op5.txt to site.pp. The script checks if the node is already present in the configuration file for Puppet. If it is not present, the node will be added to the file with the module "NRPE" included. This script is triggered by the existence of the file hosts_op5.txt via Puppet.

4.3.5 Remove nodes puppet (remove_nodes_puppet.sh)[3]

The script removes nodes from site.pp when /mnt/done.txt consists of every hostname specified in hosts_op5.txt. This means that all nodes are done and have run the module "NRPE" including the scripts. The script moves the files hosts_op5.txt and checks.txt to /mnt/archived_services_and_hosts/<filename_date_and_time_of_the_move>, and then it deletes every hostname in /mnt/done.txt. If a script was uploaded it will also be deleted. When the hostnames are deleted from /mnt/done.txt by this script, the created solution has done its job and everything should be configured and presented nicely on the GUI for the NMS.

4.4 Test results

The test results from the test suite specified under section *3.3.3 Test Suite* are presented in *Table 4.4.1* which shows the average time it took for each of the specified tests to complete. Some of the tests were not possible to perform since the automated tests do this in a flow. These are shown in *Table 4.4.1* with an asterix. *Figure 4.4.1* aims give the reader a more visual overview of the test results.

[1] Attachment *G – Add services linux*
[2] Attachment *H – Add nodes puppet*
[3] Attachment *I – Remove nodes puppet*

Test	**Man. 10 nodes (min)**	**Aut. 10 nodes (min)**	**Aut. 20 nodes (min)**
Install and configure agent	11.23	04.44	06.01
Add standard services for NRPE, PING and SSH in GUI.	07.40	*[1]	*[2]
Add one new service on nodes	10.31	04.59	04.37
Add one new service to the GUI	20.43	*[3]	*[4]
Change a parameter for a service	03.44	04.03	04.39
Total time (avg.)	**54.01**	**13.46**	**15,17**

Table 4.4.1 - Test results

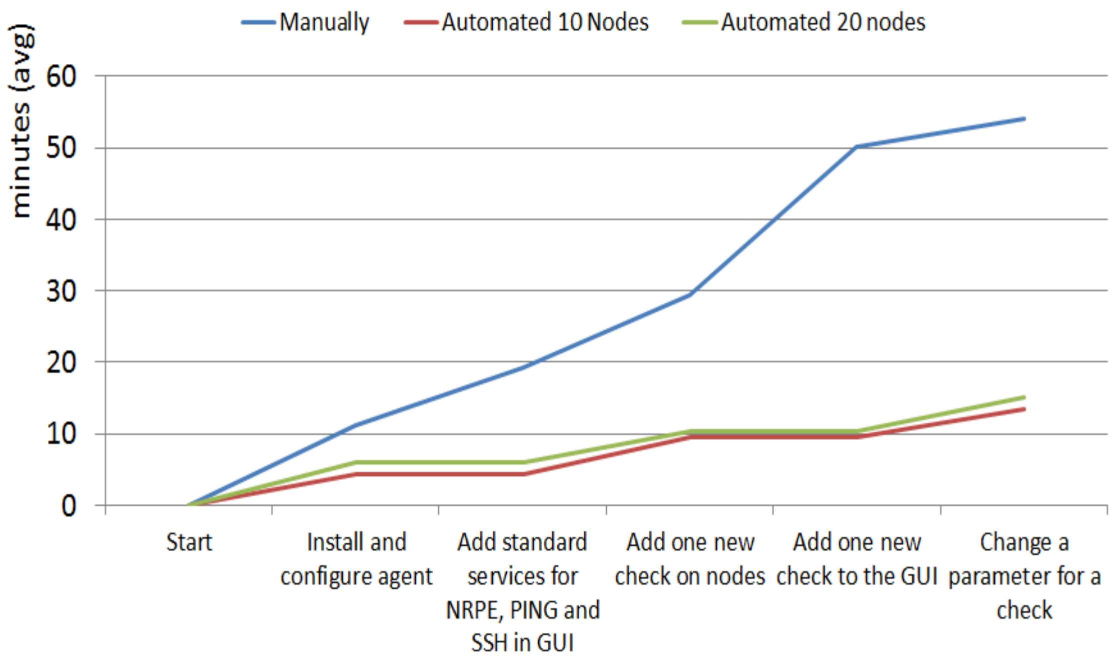


Figure 4.4.1 - Test results

[1] This step was done during the "Install and configuration agent"-step for the automated solution.
[2] This step was done during the "Install and configuration agent"-step for the automated solution.
[3] This step was done during the "Add one new check on nodes"-step for the automated solution.
[4] This step was done during the "Add one new check on nodes"-step for the automated solution.

4.5 Overview

To fully understand the relationships and dependencies between all parts of the automated solution the authors have created *Figure 4.5.1* to make this more visual to the reader. The figure will be broken down and described step by step during this section.

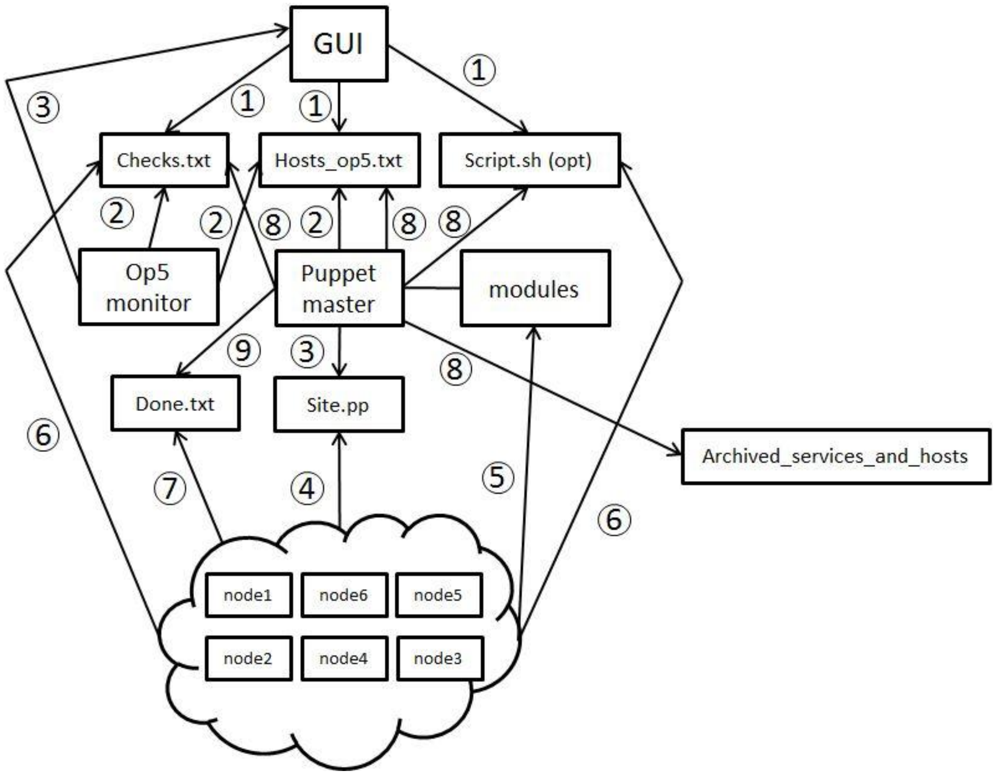


Figure 4.5.1 - Relationships and dependencies

1. *Figure 4.5.2* shows that files are uploaded to the file server.

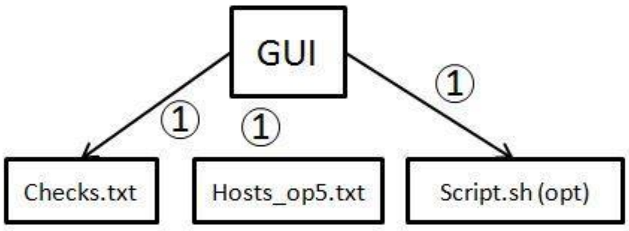


Figure 4.5.2 - Step 1

2. Puppet master and Op5 monitor notice the uploaded files via the modules on the Puppet master and the scripts on those two are being triggered. See *Figure 4.5.3* and *Code 4.5.1*.

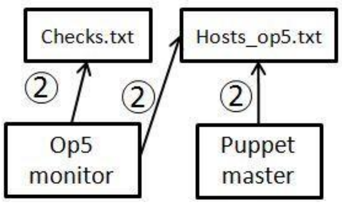


Figure 4.5.3 - Step 2

```
$script1 = '/usr/local/bin/add_nodes_puppet.sh'
$script1source = 'puppet:///modules/puppet_script/add_nodes_puppet.s
h'
```

```
class puppet_script::add_nodes inherits puppet_script
{
        file {$script1:
                source  => $script1source,
                mode    => '755'
        }

        exec {$script1:
                require => File[$script1],
                onlyif  => 'test -f /mnt/upload/hosts_op5.txt'
        }
}
```

Code 4.5.1 –Trigger scripts

3. Puppet master adds the hostnames specified in hosts_op5.txt to the site.pp as node definitions, including the module NRPE. Op5 monitor adds the hosts if they do not exist, and add services to the specified hosts in the GUI. *See Figure 4.5.4* and *Code 4.5.2.*

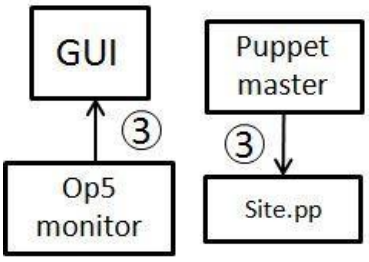


Figure 4.5.4 - Step 3

```
#If exist_value is greather then 0 it means that the node is already
 present.
if [ $exist_value -gt 0 ];
then
        echo "Node $node already exist."
#Or if exist_value is equal to 0 it means that the node is not prese
nt and should then be added to site.pp.
elif [ $exist_value -eq 0 ];
then
```

```
        echo "Node $node added."
        echo "$node_full" >> /etc/puppet/manifests/site.pp
fi
```

Code 4.5.2 – Add node definitions to site.pp

4. The nodes check via the Puppet agent if there are any definition for the specific node (in site.pp), which it is by now. *See figure 4.5.5.*

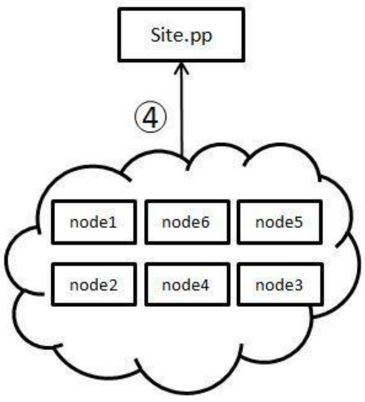


Figure 4.5.5 - Step 4

5. The nodes pull down the configurations from the module "NRPE" and execute it. See *Figure 4.5.6.*

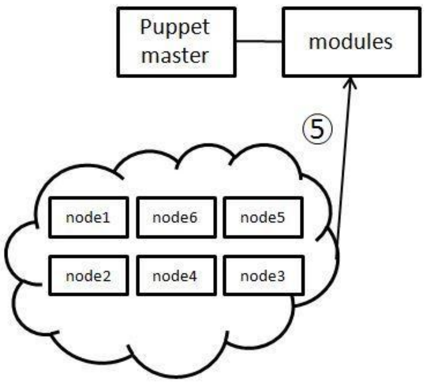


Figure 4.5.6 - Step 5

6. *Figure 4.5.7* shows the scripts to be executed on the nodes. It uses checks.txt to add those services to the node[1]. It copies the script file and makes it executable if a script file is uploaded, see *Code 4.5.3*.

[1] Attachment *E – Add check on nodes*

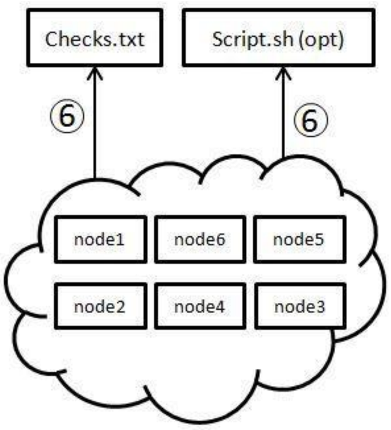


Figure 4.5.7 - Step 6

```
#Add new script-file to local storage for the NRPE-scripts.
                script_name=`ls /mnt/scripts/`
                if [ -z ”$script_name” ];
                then
                        echo “No script to upload.”
                else
                        cp “/mnt/scripts/$script_name” /opt/plugins/

                        chmod +x /opt/plugins/”$script_name”
fi
```

Code 4.5.3 – Copy script file and makes it executable.

7. When a node has finished the script the node echoes its hostname to /mnt/done.txt. See *Figure 4.5.8* and line *97-133.*[1]

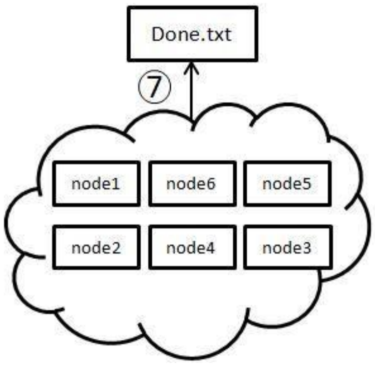


Figure 4.5.8 - Step 7

[1] Attachment *E – Add check on nodes*

8. When done.txt has all the same hostnames as in hosts_op5.txt the Puppet master moves checks.txt and hosts_op5.txt. It also deletes the script file. The filename of checks.txt and hosts_op5.txt are added with the current date and time. See *Figure 4.5.9*, *Figure 4.5.10* and Attachment *I – Remove nodes puppet*.

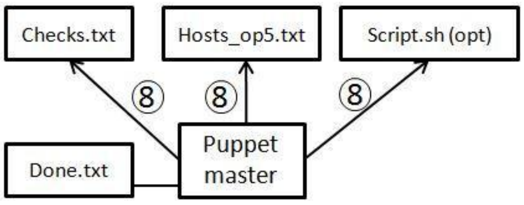


Figure 4.5.9 - Step 8

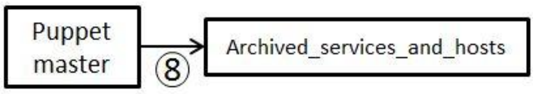


Figure 4.5.10 - Step 8

9. When those files are removed, the Puppet master clears /mnt/done.txt as can be seen in *Figure 4.5.10* and *Code 4.5.4*.

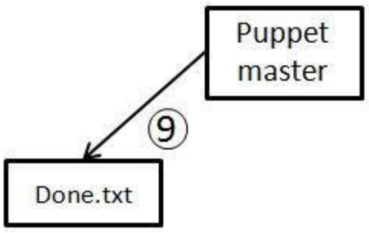


Figure 4.5.10 - Step 9

```
# For every node in done.txt, remove it from site.pp and then done.txt.
        while read node
        do
              node_full="node '$node' { include nrpe }"
              sed -i "/$node_full/d" /etc/puppet/manifests/site.pp
              sed -i "/$node/d" /mnt/done.txt
        done < /mnt/done.txt
```

Code 4.5.4 – Remove nodes from site.pp.

5. Discussion

The discussion section presents the authors analysis regarding the results and the implementation of this paper. This section consists of discussions regarding the WEB GUI, created scripts, Puppet with its modules, the test results and other related discussions.

5.1 WEB GUI

The widget is, as stated in section *4.1 WEB GUI*, written in HTML and PHP. The upload function consist of a basic PHP upload script, modified to fit this paper's IT environment.

Since the widget is created in HTML and PHP it should not be any problem to migrate the widget to another NMS and its GUI. There are some parameters that need configuration to fully function if the widget is to be placed in a new environment, such as path for pictures and to the file server. Another solution would be to integrate Ninja because the widget is tested and fully functional during these circumstances. The authors are aware of the problem concerning no error handling when uploading files. It is by the time of writing possible to upload a script without specifying any hostnames nor services, or vice versa. This means if the user has uploaded several scripts and then uploads a checks.txt containing these scripts commands and parameters, it will probably not work because the script on the nodes cannot fetch a single filename in the script path.

By installing the widget, the NMS enables a way to interact with monitored nodes in a way that was not possible before. A possible security risk occurs if an intruder gets access to the NMS. The intruder could then upload a file via the "Upload new script (opt)" field and specify a number of hosts in the "Hostnames" field and thereby deploy the script to the specified nodes. This requires that the intruder is aware of some hostnames in the environment to which the script should be uploaded to.

The configuration to present the "work in progress"-text could indeed be configured in a more user friendly and correct way. The current solution requires the user to update the widget page. The widget echoes the file /mnt/done.txt. At first it will echo nothing, because the file is empty. But when nodes have run their modules and script they echo their hostnames to done.txt, which is then echoed to the widget page. When all hostnames are present in done.txt the Puppet master will clear the file, which will result in an empty "work in progress" once again on the widget page. By now everything should be converged.

A more proper solution for this would be to use some kind of visualization to make it more user friendly and understanding. A progress bar which loads depending on amount of finished hosts, and a text which updates and writes the current finished host(s).

By the time of writing the files to be uploaded in the "Hostnames"- and "Add and / or change service"-field must be named as in *Table 4.1.1 – Web GUI.*

5.2 Puppet

The Puppet agent must be installed either manually or by some other deployment software. In the scenario presented in section *3.3.1 Test Environment*, the authors used a template to deploy the new machines. On the template the puppet agent was installed, all that had to be done was to accept the certificate request on the Puppet master after cloning. The automated solution does not automatically accept requested certificates by the Puppet nodes. The reason for not implementing a function to accept requested certificates is because Puppet labs do not recommend doing so. [32] If you would create a function to automatically accept requested certificates you would not have the same control of the accepted nodes. Regarding automatic solutions and for information, please read Puppet Labs page about it in [32].

One of the limitations of Puppet is that the ability to "push" updates to the nodes is considered deprecated and has been withdrawn [26]. Instead the user is limited to use "pull" from the nodes. By default this update occurs every 30 minutes. By specifying the *runinterval* parameter in puppet.conf, it is possible to update more frequently. [23] In the test scenario of this paper a *runinterval* of 120 seconds was configured.[1] This value should be adjusted to the specific environment. In a large environment it could be possible that if too many of the nodes connect to the Puppet master at the same time, it could be overloaded in what could be considered a Distributed Denial of Service (DDoS).

If the Puppet master is running slowly, it should be considered to configure a higher *runinterval*. If the *runinterval* is changed to a lower value on the Puppet master, NMS and nodes, the time to complete the automated solution is most likely lower than what the results in *Table 4.4.1 - Test results* and *Figure 4.4.1 - Test results* presents. This is due to the fact that the Puppet master will trigger faster on the uploaded file, and so will the NMS. The nodes will check more frequently if a node definition is available in site.pp and thereby run the module "NRPE" earlier than during the tests in this paper.

[1] Attachment *J - Puppet code for runinterval and usecacheonfailure*

There are some static configurations in the puppet modules which should be removed and replaced by variables. The modules would be more user friendly if variables were used. But due to time constraints the authors of this paper did not have the time to modify it before handling in this paper. However, the modules are available at Github[1], and may be updated.

The authors encountered a problem regarding caching of the Puppet configuration. While adding some services for only a few nodes, for testing purpose, the remaining nodes ran their old scripts which had been cached. They triggered due to the files were uploaded, and because they did not find any definition for them in site.pp. The solution to this was to disable Puppet to use the cache on a failure of finding a node definition [33]. This was done by a class for all nodes by defining a "node default" which applies to all nodes.[2]

There are some security issues to be concerned about regarding the automated solution for Puppet. There is no control on what the script contains that are executed on every node, pulled from the Puppet master's modules. If an intruder manage to get into the Puppet master and is able to exchange the scripts it could mean a large security risk.

5.3 Scripts

The authors have created five scripts during the work of this paper. Each script is presented in section *4.3 Scripts*. Each script is triggered by Puppet if a state is achieved, mostly if a file exists. This combination is the key to the automated solution. If the files containing the services and the hostnames are uploaded, the Puppet agent for the NMS triggers the script to be run. The scripts use the Nacoma API to modify configuration files on the NMS.

There are some static configurations in the scripts which need to be changed to variables before applying the solution in a new environment. But due to time constraints the author's did not have time to correct it. But the scripts are available at Github[1], and may thereby be updated later on.

5.4 Test results

It is a very time consuming task to perform these activities manually, as can be seen in *Figure 4.4.1 – Test results*. But it also increases the risk of human errors. Mistypes and forgotten activities, that leads to errors and troubleshooting. In the baseline test, performed by the authors, a lot of small errors had to be corrected which was added to the time. The test results presented for the baseline test is the actual time spent to perform the specified

[1] https://github.com/jesajah/rosi
[2] Attachment *J - Puppet code for runinterval and usecacheonfailure*

activities. For the automated tests the duration was not only lower, the actual time that involved human interaction was almost non-existent. The only steps the authors did were the once you are supposed to do when using this paper's solution: choose two or three files to upload through the GUI and then click "submit". After that the only thing to do is to wait.

The full durations of the tests were remarkably lower than the manual test. In two of the performed tests ten nodes were used and as these tests scale up, there should be a greater risk of human errors during a manual performance and the time spent should increase linearly [27]. The time spent is not linear when the automated solution is used, as *Figure 4.4.1 – Test results* and *Table 4.4.1 – Test results*, clearly shows. The scalability for the automated solution is good according to this result. But, as stated before, due to hardware shortage the authors of this paper have not tried the automated solution with a larger amount of nodes than 20 to see if the scalability is different with hundreds of nodes.

The time-span varied between ~02:56 to ~06:34 for the different steps, presented in *Table 4.4.1 – Test results*. This time-span depends on the *runinterval* for the Puppet agent on the nodes, NMS and the Puppet master. Let's say the user uploads the files ten seconds after the Puppet master's Puppet agent has executed. The user will then need to wait approximately 1 minute and 50 seconds before the Puppet master will execute again and do the first step on the automated solution. This given that the *runinterval* on the Puppet agents are set to 120 seconds as in the test environment of this paper.

The nodes may run when the Puppet master has done the first step, which imply adding the node definitions to site.pp. But this may take up to one *runinterval*. Let's say the Puppet master adds the node definitions to the site.pp just after a node checks whether there are any Puppet configurations for the node or not. Which it will not be because the Puppet master has not added any of the nodes to the site.pp yet. The node has to wait for almost two minutes before noticing the Puppet configurations to be run. Imagine this scenario for 100 nodes. The more nodes added, the more likely a node times the *runinterval* badly and the more likely the time lands in the upper time-span presented in the previous paragraph.

5.5 Other discussions

The solution has not been packed for distribution to other systems by the time of writing. Some of the variables in the code are "hard coded" for the test environment and should be changed for a more general approach. But as mentioned in section *5.3 Scripts* the code is published on Github[1] and may thereby be updated later on.

A problem occurred while the authors made the tests for the automated solution. The checks.txt and hosts_op5.txt were corrupted if they were opened on a Windows machine. The solution to the problem was to only open and modify the files in Linux. A possible reason of the problem is that newlines are represented by "^M" in Windows and "\n" in Unix like systems. There are programs like [31] whose task is to convert text files from Windows to Unix style. But none of which the authors have implemented or tested to suit the presented solution.

[1] https://github.com/jesajah/rosi

6. Conclusion

This paper presents a way to simplify deployment and management of agents for Network Monitoring Systems on Linux distributions. To resubmit the purpose and the problem presented in this paper:

"How is it possible to simplify management and deployment of agents and services on monitored nodes in a mixed IT infrastructure in a reasonable time?"

This paper began with an introduction of the problem, the target audience and the public good it would have on system administrators in the business. The authors of this paper then presented a method on how they think this is possible. By integrating a Content Management System with the Network Monitoring System and develop an easy to use Web based Graphical User Interface the authors present *one* way to simplify management and deployment of agents. The advantages and disadvantages with an automated solution compared to manual administration are discussed. What the reader considers reasonable time is of course very individual, but is here specified as the time it would take for a system administrator to manually perform the chosen activities. These manual activities are presented as a baseline for comparison with the automated solution. The presented solution has been made more easy to use through an integrated widget in the Network Monitoring System. The solution consists of Puppet, a Content Management System, with some modules with different tasks.

Imagine an IT environment with 650 nodes, everything from webservers to a large clustered render farm. No one would like to manually install a Network Monitoring System agent on all those nodes. A system administrator would probably script a solution to install the Network Monitoring System agent on all nodes. Another system administrator might consider using a Content Management System like Puppet. Either way would probably take a lot of time to complete. When the nodes have the agent installed the Network Monitoring System still needs to be configured to monitor all those nodes, either manually add all nodes via the graphical user interface or the command line, or by using an Application Programming Interface (API), if it is available for the chosen Network Monitoring System. These solutions would most probably take a lot of time to finish. And what if all these nodes need a new service? The administrator then needs to connect to each node and add the service and add it to all nodes via the Network Monitor System graphical user interface or API.

With the solution presented in this paper deployment, configuration and management of agents for Network Monitoring System are a lot more user friendly and it takes only a few minutes to distribute the agents and services, including adding the nodes to the Network Monitoring System's graphical user interface. By combining Puppet with several of scripts created by the authors of this paper the deployment and management of Network Monitoring System agents are made in a reasonable time. The solution does also include configuration on the Network Monitoring System. The solution adds hosts and the services to the Network Monitoring System. It also adds, or changes, the services on the nodes (hosts). The solution has been integrated with the Network Monitoring System Op5's graphical user interface via a widget to simplify the usability of the solution.

6.1 Further research

The authors believe that a deeper analysis of the security aspects would be necessary before implementing the solution in a production environment.

This paper only treats management and distribution of Network Monitoring Agents on Centos. The authors hope that further researchers investigate the possibility to port the solution to other Linux distributions and Windows platforms. The authors have discussed the most vital configurations that need to be changed before implementing the solution on other Linux distributions. As mentioned in earlier sections, the code may be updated by the use of Github.[1]

[1] https://github.com/jesajah/rosi

References

[1] C. Armas, “Puppet: Ruby-based Server Management Automation Suite”, 2010, [Online]. Available: http://www.infoq.com/news/2010/02/puppet-25 [Accessed: May 24, 2013]

[2] C. H. Richard, "Network management with Nagios," *Linux J.*, vol. 2003, p. 3, 2003.

[3] F. Engel, K. S. Jones, K. Robertson, D. M. Thompson, and G. White, "Network monitoring," ed: Google Patents, 2000

[4] J. G. Ochin, "NETWORK MONITORING SYSTEM TOOLS: AN EXPLORATORY APPROACH.", *UACEE International Journal of Advances in Computer Networks and Security*, Manav Rachna International University

[5] Nagios, [Online]. Available: http://http:/www.nagios.com. [Accessed: April 8, 2013]

[6] Op5 AB, “Company History”, [Online]. Available:http://www.op5.com/about/about-op5/company-history/. [Accessed: April 8, 2013]

[7] Op5 AB, “Nagios Based Monitoring", [Online]. Available: http://www.op5.com/op5-features/nagios-based-monitoring/ [Accessed: April 8, 2013]

[8] Anonymous, "Cisco Global Cloud Index: Forecast and Methodology, 2011–2016," ed: Cisco Systems, 2012.

[9] T. Delaet and W. Joosen, "Survey of configuration management tools," 2007.

[10] Puppet-Labs, "Configuration Management", [Online]. Available: https://puppetlabs.com/solutions/configuration-management/ [Accessed: April 8, 2013]

[11] Op5 AB, [Online]. Available: http://www.op5.com/release-notes/op5-monitor-6-0-release-notes/ [Accessed: May 22, 2013]

[12] Önnberg F. Software Configuration Management : A comparison of Chef, CFEngine and Puppet. 2012.

[13] Puppet Labs, “Supported Platforms”, [Online]. http://docs.puppetlabs.com/guides/platforms.html [Accessed: May 21, 2013]

[14] VMware, “”VMware vSphere Hypervisor”, [Online]. Available: http://www.vmware.com/products/vsphere-hypervisor/overview.html [Accessed: May 21, 2013]

[15] J. Turnbull and J. McCune, Pro Puppet: Apress, 2011.

[16] A. Bryman, *Social research methods*, 3. ed. Oxford: Oxford University Press, 2008. ch. 25

[17] B. Perens, "The open source definition," Open sources: voices from the open source revolution, pp. 171-85, 1999

[18] CentOS Doc, “Chapter 18. Network File System (NFS)”, [Online]. Available: http://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-nfs.html [Accessed: May 21, 2013]

[19] V. Hendrix, D. Benjamin, and Y. Yao, "Scientific Cluster Deployment and Recovery–Using puppet to simplify cluster management," in Journal of Physics: Conference Series, 2012, p. 042027

[20] C. Issariyapat, P. Pongpaibool, S. Mongkolluksame, and K. Meesublak, "Using Nagios as a groundwork for developing a better network monitoring system," in Technology Management for Emerging Technologies (PICMET), 2012 Proceedings of PICMET '12:, 2012, pp. 2771-7

[21] GNU, “GNU Bash”, [Online]. Available: http://www.gnu.org/software/bash/bash.html [Accessed: May 21, 2013]

[22] K. Adam, "Puppet and nagios: a roadmap to advanced configuration," Linux J., vol. 2012, p. 3, 2012

[23] Puppet labs, “Overview”, [Online]. Available: http://docs.puppetlabs.com/pe/2.8/puppet_overview.html [Accessed: May 21, 2013]

[24] P. Schaffner, “Yum”, 2011, [Online]. Available: http://wiki.centos.org/PackageManagement/Yum [Accessed: May

21, 2013]

[25] Google Patent, “WO 2009044138 A2”, [Online]. Available: http://www.google.com/patents/WO2009044138A2?cl=en [Accessed: May 22, 2013]

[26] Eric Sorenson, “Bug #15735”, [Online]. Available: http://links.puppetlabs.com/puppet-kick-deprecation [Accessed: May 21, 2013]

[27] D. A. Norman, "Design rules based on analyses of human error," Commun. ACM, vol. 26, pp. 254-8, 1983.

[28] Op5 AB, “Nacoma - Nagios Configuration Manager”, [Online]. Available: http://www.op5.org/community/projects/nacoma [Accessed: May 21, 2013]

[29] Op5 AB, “Learning - Modules 1” [Online]. Available: http://docs.puppetlabs.com/learning/modules1.html [Accessed: May 21, 2013]

[30] Op5 AB, “Learning - Variables, Conditionals and Facts” [Online]. Available: http://docs.puppetlabs.com/learning/variables.html [Accessed: May 21, 2013]l

[31] M. G. Sobell, A practical guide to Linux: Addison-Wesley Longman Publishing Co., Inc., 1997. p. 55

[32] Puppet labs, “Certificates and Security”, [Online]. Available: http://projects.puppetlabs.com/projects/1/wiki/certificates_and_security [Accessed: May 21, 2013]

[33] Puppet labs, “Configuration Reference - usecacheonfailure”, [Online]. Available: http://docs.puppetlabs.com/references/latest/configuration.html#usecacheonfailure [Accessed: May 28, 2013]

[34] Op5 AB, “Ninja”, [Online]. Available: http://www.op5.org/community/plugin-inventory/op5-projects/ninja [Accessed: May 21, 2013]

[35] Op5 AB, “Merlin”, [Online]. Available: http://www.op5.org/community/plugin-inventory/op5-projects/merlin [Accessed: May 22, 2013]

[36] Op5 AB, “User Manual op5 NRPE 2.7.0”, 2006, [Online].

Available:
http://www.op5.com/manuals/extras/op5_NRPE_2.7_manual.pdf
[Accessed: May 28, 2013]

[37] A. M. Graziano and M. L. Raulin. (2010). *Research methods,* pp. 30-1

Attachments

Contents

A – WEB GUI ____ i
B - Manifest for puppet_script ____ v
C – Manifest for NRPE ____ vi
D - Manifest for monitor_script ____ ix
E – Add check on nodes ____ x
F - Add hosts ____ xiii
G - Add services linux ____ xiv
H - Add nodes puppet ____ xvi
I - Remove nodes puppet ____ xvii
J - Puppet code for runinterval and usecacheonfailure ____ xix

A – WEB GUI

```
<html>
<head>
<style type="text/css">
.tooltip-wrap {
  position: relative;
}
.tooltip-wrap .tooltip-content {
  display: none;
  position: absolute;
  top: 15%;
  left: 5%;
  right: 5%;
  background-color: #F0F0F0;
  padding: .5em;
}
.tooltip-wrap:active .tooltip-content {
  display: block;
}

.image-lnu {
 position: absolute;
 top: 85%;
 left: 85%;
}
</style>
</head>
<body>

<!-- Information to the viewer -->
<p>This widget is used to add new hosts and to add services to new and / or existing
 nodes.
<br><br>
<div class="image-lnu">
<img src="http://monitor.rosi.local/lnu.jpeg" />
</div>
<div class="tooltip-wrap">
  <img src="http://monitor.rosi.local/question.jpg" alt="Some Image" width="35" heig
ht="35" />
 <div class="tooltip-content">
    <p> Example file:<br><br>
        linuxnode1.example.com<br>
        linuxnode2.example.com<br>
        linuxserver.example.com<br>
        <br>
(.txt and no larger than 20 KB)
</p>
  </div>
</div>

<!-- Upload hostname and checks files -->
<form action="" method="post"
        enctype="multipart/form-data">
        <label for="hostname">Hostnames:</label><br>
        <input type="file" name="hostname" id="hostname"><br>

    <div class="tooltip-wrap">
 <img src="http://monitor.rosi.local/question.jpg" alt="Some Image" width="35" height="35" />
 <div class="tooltip-content">
  <p> <p>Example file:<br><br>
    command[users]=/opt/plugins/check_users -w 5 -c 10<br>
    command[load]=/opt/plugins/check_load -w 15,10,5 -c 30,25,20<br>
    command[swap]=/opt/plugins/check_swap -w 20% -c 10%<br>
    <br>
    (.txt and no larger than 20 KB)<br>
    <br>
  </p>
 </div>
</div>
    <label for="service">Add and/or change service:</label><br>
    <input type="file" name="service" id="service"><br>

    <div class="tooltip-wrap">
 <img src="http://monitor.rosi.local/question.jpg" alt="Some Image" width="35" height="35" />
 <div class="tooltip-content">
  <p>Choose the script file to be uploaded. The script's command should be included in the file above.<br>
    <br>
    (Script files and no larger than 20KB)
  </p>
 </div>
</div>
    <label for="script">Upload new script(opt):</label><br>
    <input type="file" name="script" id="script"><br>


<p> Please note that the change may take up 7-8 minutes to complete.</p>

<input type="submit" name="submit" value="Submit">
<br>

<?php
error_reporting (E_ALL ^ E_NOTICE);
if (isset($_POST['submit'])){
//Hostname begins
    if ((($_FILES["hostname"]["type"] == "text/plain") && ($_FILES["hostname"]["size"]< 20000))){
        if ($_FILES["hostname"]["error"] > 0) {
            echo "Error: " . $_FILES["hostname"]["error"] . "<br>";

        }
        else {
            echo "Upload: " . $_FILES["hostname"]["name"] . "<br>";
            echo "Stored in: " . ($_FILES["hostname"]['/mnt/upload/']);

               }
           if (file_exists("/mnt/upload/" . $_FILES["hostname"]["name"])) {
               echo $_FILES["hostname"]["name"] . " already exists." . "<br>";
           }
           else {
               move_uploaded_file($_FILES["hostname"]["tmp_name"],
               "/mnt/upload/" . $_FILES["hostname"]["name"]);
               echo "Stored in: " . "/mnt/upload/" . $_FILES["hostname"]["name"] . "<br>";
               }
       }
       else {
            echo "Invalid file" . "<br>";
        }

    //Service begins
        if ((($_FILES["service"]["type"] == "text/plain") && ($_FILES["service"]["size"]< 20000))){
            if ($_FILES["service"]["error"] > 0) {
                echo "Error: " . $_FILES["service"]["error"] . "<br>";

            }
            else {
                echo "Upload: " . $_FILES["service"]["name"] . "<br>";

                echo "Stored in: " . ($_FILES["service"]['/mnt/upload/']);
            }
            if (file_exists("/mnt/upload/" . $_FILES["service"]["name"])) {
                echo $_FILES["service"]["name"] . " already exists. " . "<br>";
            }
            else {
                move_uploaded_file($_FILES["service"]["tmp_name"],
                "/mnt/upload/" . $_FILES["service"]["name"]);
                echo "Stored in: " . "/mnt/upload/" . $_FILES["service"]["name"] . "<br>";
            }
        }
        else {
            echo "Invalid file" . "<br>";
        }

    //Script begins
        if ((($_FILES["script"]["type"] == "application/octet-stream"))
            && ($_FILES["script"]["size"]< 20000)){

            if ($_FILES["script"]["error"] > 0) {
                echo "Error: " . $_FILES["script"]["error"] . "<br>";

            }
            else {
                echo "Upload: " . $_FILES["script"]["name"] . "<br>";

                echo "Stored in: " . ($_FILES["script"]['/mnt/scripts/']);
            }
            if (file_exists("/mnt/scripts/" . $_FILES["script"]["name"])) {
                echo $_FILES["script"]["name"] . " already exists. " . "<br>";
            }
            else {
                move_uploaded_file($_FILES["script"]["tmp_name"],
                "/mnt/scripts/" . $_FILES["script"]["name"]);
                echo "Stored in: " . "/mnt/scripts/" . $_FILES["script"]["name"] . "<br>";
            }
        }
        else {
            echo "Invalid file" . "<br>";
        }
    }

    echo "For progress, update the page:" . "<br>";
    $filestring = file_get_contents('/mnt/done.txt');
    echo nl2br( htmlspecialchars($filestring) );
```

```

?>
</form>

<br>
<br>
© Simon Blixt & Robin Jonsson.
</body>
</html>
```

B - Manifest for puppet_script

```
## == pupppet_script

class puppet_script {

$script1 = '/usr/local/bin/add_nodes_puppet.sh'
$script1source = 'puppet:///modules/puppet_script/add_nodes_puppet.sh'
$script2 = '/usr/local/bin/remove_nodes_puppet.sh'
$script2source = 'puppet:///modules/puppet_script/remove_nodes_puppet.sh'

Exec {
        path    => ['/sbin', '/bin','/usr/sbin', '/usr/bin' ]
}

#Define in which order the subclasses should be executed

        class {'puppet_script::add_nodes': } ->
        class {'puppet_script::remove_nodes': }
}

#Subclass add_nodes which checks if /mnt/upload/hosts_op5.txt exist. If it exists the script add_nodes_puppet.sh is executed on the Puppet master.
class puppet_script::add_nodes inherits puppet_script
{
        file {$script1:
                source  => $script1source,
                mode    => '755'
        }

        exec {$script1:
                require => File[$script1],
                onlyif  => 'test -f /mnt/upload/hosts_op5.txt'
        }
}

#Subclass remove_nodes which checks if /mnt/test.txt (points on /mnt/done.txt). If it changes the script remove_nodes_puppet.sh is executed on the Puppet master.
class puppet_script::remove_nodes inherits puppet_script
{
        file {$script2:
                source  => $script2source,
                mode    => '755'
        }

        file {'/mnt/test.txt':
                mode    => '766',
                source  => '/mnt/done.txt'
        }

        exec {$script2:
                subscribe       => File['/mnt/test.txt'],
                refreshonly     => true
        }
}
```

C – Manifest for NRPE

```
## == nrpe

#Main class for NRPE, define variables etc
class nrpe (

)

#Defines how the subclasses should be executed

{
        class {'nrpe::mount': } ->
        class {'nrpe::install': } ->
        class {'nrpe::files': } ->
        class {'nrpe::service': } ->
        class {'nrpe::run_script': }
}

#Ensures that the fileserver is mounted.

 class nrpe::mount {
        Exec {
                path    => ['/sbin', '/bin','/usr/sbin', '/usr/bin' ]
        }
        mount {'/mnt':
                device  => 'fileserver.rosi.local:/share',
                fstype  => 'nfs',
                ensure  => 'mounted',
                options => 'defaults',
                atboot  => 'false'
        }
}

#Makes sure that the dependencies of NRPE is installed, and then installs NRPE from the packages located on the fileserver.

 class nrpe::install {

       package {'gnutls':
                ensure  => installed,
        }

        package {'mysql':
                ensure  => installed,
                require => package['gnutls']
        }

        package {'postgresql':
                ensure  => installed,
                require => package['mysql']
        }

        package {'nrpe_nagiosplugins-2.13.1-release.x86_64':
                ensure  => installed,
                provider => rpm,
                source  => '/mnt/packages/nrpe-2.13-nagios_plugins-1.4.15-CentOS_6-2.13.1_x86_64.rpm',
                require => package['postgresql']
        }
}

#Disabling SElinux. Otherwise NRPE can't communicate with the NMS. Should add an exception instead later on.
```

```

 class nrpe::files {
        Exec {
                path    => ['/sbin', '/bin','/usr/sbin', '/usr/bin' ]
        }

        exec {'setenforce 0':
                onlyif  => 'grep -c SELINUX=enforcing /etc/selinux/config',
                before  => file_line['remove_line']
        }

        file_line {'remove_line':
                path    => '/etc/selinux/config',
                line    => 'SELINUX=enforcing',
                ensure  => absent
        }

        file_line {'add_selinux_disabled':
                path    => '/etc/selinux/config',
                line    => 'SELINUX=disabled',
                require => file_line['remove_line']
        }

}

#Edit the NRPE conf to allow the op5 monitor to communicate with the node. Restart the service when the config-file is changed.

 class nrpe::service {

        file_line {'remove_hosts':
                path    => '/etc/nrpe.conf',
                line    => 'allowed_hosts=127.0.0.1',
                ensure  => absent,
                before  => file_line['allowed_hosts']
        }

        file_line {'allowed_hosts':
                path    => '/etc/nrpe.conf',
                line    => 'allowed_hosts=127.0.0.1,139.139.139.4'
        }

              file {'/etc/nrpe.conf':
                      source   => 'puppet:///files/nrpe.conf'
              }

              exec {'service nrpe restart':
                      path            => ['/sbin', '/bin','/usr/sbin', '/usr/bin' ],

                      subscribe       => File['/etc/nrpe.conf'],
                      refreshonly     => true
              }
     }

     # Executes the script add_check_on_nodes.sh if the file /mnt/upload/checks.txt exists.
     class nrpe::run_script {

     $script1 = '/usr/local/bin/add_check_on_nodes.sh'
     $script1source = 'puppet:///modules/nrpe/add_check_on_nodes.sh'

              Exec {
                      path    => ['/sbin', '/bin','/usr/sbin', '/usr/bin' ]
              }

              file {$script1:
                      source   => $script1source,
                      mode     => '755'
               }

              exec {$script1:
                      require         => File[$script1],
                      onlyif          => 'test -f /mnt/upload/checks.txt'
              }
     }
```

D - Manifest for monitor_script

```
 ## == monitor_script

# Main class for monitor_script, define variables etc.

class monitor_script {

$script1 = '/usr/local/bin/add_hosts.sh'
$script1source = 'puppet:///modules/monitor_script/add_hosts.sh'
$script2 = '/usr/local/bin/add_services_linux.sh'
$script2source = 'puppet:///modules/monitor_script/add_services_linux.sh'

#Define in which order the subclasses should be executed

        class {'monitor_script::run_hosts': } ->
        class {'monitor_script::run_checks': }
#       class {'monitor_script::unmount': }
}

#Sending the script "add_hosts.sh" to the monitor and execute it if the file /mnt/upload/hosts_op5.txt exists.

 class monitor_script::run_hosts inherits monitor_script {

        Exec {
                path    => ['/sbin', '/bin','/usr/sbin', '/usr/bin' ]
        }

        file {$script1:
                source  => $script1source,
                mode    => '755'
        }

        exec {$script1:
                require => File[$script1],
                onlyif  => 'test -f /mnt/upload/hosts_op5.txt'
        }
 }

#Sending the script "add_services_linux.sh" to the monitor and execute it if the file /mnt/upload/checks.txt exists.

 class monitor_script::run_checks inherits monitor_script {

        Exec {
                path    => ['/sbin', '/bin','/usr/sbin', '/usr/bin' ]
        }

        file {$script2:
                source  => $script2source,
                mode    => '755'
        }

        exec {$script2:
                require         => File[$script2],
                onlyif          => 'test -f /mnt/upload/checks.txt'
        }
 }
```

E – Add check on nodes

```bash
#!/bin/bash
#Script to add checks on nodes(linux)

#path to the monitors check-file.
path_op5_commands="/etc/nrpe.d/op5_commands.cfg"

#Read the new checks
while read new_check
do

#Variable to check  if new check is present, changed, or new.
exists_nr=0
#Variable to check if a restart should occur or not.
restart_value=0

    #Strips out the new check's command-name
    new_check_command=$(echo "$new_check" | awk -F] '{print $1}'|awk -F[ '{print $2}')

        #Read the existing checks
        while read existing_check
        do
            #Strips out the existing check command-name
            existing_check_command=$(echo "$existing_check" | awk -F] '{print $1}'|awk -F[ '{print $2}')

            #If new check command-name equals existing check command-name..
            if [ "$new_check_command" == "$existing_check_command" ];
            then
                #..check if the whole new check is equal to the existing.
                if [ "$new_check" == "$existing_check" ];
                then
                    #If it is, increase $exists_nr with 1.
                    exists_nr=$(($exists_nr + 1 ))
                #If not, it means the new check has different values.
                else
                    #Therefore, decrease $exists_nr with 1..
                    exists_nr=$(($exists_nr - 1 ))
                    #.. and sets the $existing_check to $delete_check..
                    delete_check=$existing_check_command
                    #.. and $new_check to $add_check
                    add_check=$new_check
                fi
            else
                #If new checks command is not equal to existing command (which means the new check does not exist with different parameters)..
                #.. sets $exists_nr with plus 0.
                exists_nr=$(($exists_nr + 0 ))
                #Sets $add_check to $new_check
                add_check=$new_check

            fi
        done < /etc/nrpe.d/op5_commands.cfg

    #If $exists_nr is greater then 0..
    if [ $exists_nr -gt 0 ];
    then
        #Print out that the check is already present.
        echo "The check, $new_check, is already present."
    #Or if $exists_nr is lower then 0..
    elif [ $exists_nr -lt 0 ];
    then
        #Print out that the check is present but has different values.
        echo "The check $new_check is present but has different values. Deleting old and adding new."
        #And delete the line which is the existing one, with old values.
        sed -i "/$delete_check/ d" $path_op5_commands
        #And add the new one, with the new values.
        $(echo "$add_check" >> "$path_op5_commands" )
        #Increase $restart_value with 1.
        restart_value=$(($restart_value + 1 ))
    #Or if $exists_nr is equal to 0..
    elif [ $exists_nr -eq 0 ];
    then
        echo "The check $new_check has been added"
        #.. the new check is added.
        $(echo "$add_check" >> "$path_op5_commands")
        #And the $restart_value is increased with 1.
        restart_value=$(($restart_value + 1 ))
        #Add new script-file to local storage for the NRPE-scripts.
        script_name=`ls /mnt/scripts/`
        if [ -z "$script_name" ];
        then
            echo "No script to upload."
        else
            cp "/mnt/scripts/$script_name" /opt/plugins/
            chmod +x /opt/plugins/"$script_name"
        fi
    else
        echo "Unknown error. $exist_nr"
    fi
done < /mnt/upload/checks.txt

#Check if the $restart_value is greater then 0..
if [ $restart_value -gt 0 ];
then
    #.. if so, restart nrpe. This means that a new check has been added, or an old one has been modified.
    service nrpe restart
fi

#If OP5_Commands = uploaded file; then report done.
counter=$(wc -l < /mnt/upload/checks.txt )
exist_value=0
    while read new_check
    do

        while read existing_check
        do
            if [ "$new_check" == "$existing_check" ];
            then
                exist_value=$(($exist_value + 1 ))
            else
                exist_value=$(($exist_value + 0 ))
            fi

        done < /etc/nrpe.d/op5_commands.cfg
    done < /mnt/upload/checks.txt

    if [ $exist_value -eq $counter ];
    then
        if [ -f /mnt/done.txt ];
        then
            hostname=$(hostname -f)
            test=`cat /mnt/done.txt | grep "$hostname"`
            if [ "$hostname" == "$test" ];
            then
                echo "Hostname exist in /mnt/done.txt, nothing to do."
            else
                echo "All checks are present."
                echo "$hostname" >> /mnt/done.txt
            fi
        else
            echo "ERROR!The file /mnt/done.txt does not exist."
        fi
    else
        echo "ERROR! All checks have not been added."
    fi
```

F - Add hosts

```bash
#!/bin/bash
#
#Script to add hosts to op5 via CLI
unset $hostnames
unset $hostname
#Read the file with the new hosts and saves it in a variable.
hostnames=`cat /mnt/upload/hosts_op5.txt`

#Read the file with the existing hosts and saves it in a variable.
existing_hostnames=`cat /opt/monitor/etc/hosts.cfg | grep host_name | awk '{print $2}'`

#Loops all hostnames in $hostnames and puts one after one in $hostname
for hostname in $hostnames
do

#A value to indicate if changes have occured.
existing_value=0
        #For each existing host..
        for existing_hostname in $existing_hostnames
        do
                #.. check if it is the same hostname as the new host.
                if [ "$existing_hostname" == "$hostname" ];
                then
                        #If it is, add  + 1 to the existing_value.
                        existing_value=$(($existing_value + 1))
                else
                        #If it is not, add nothing to the existing_value.
                        existing_value=$(($existing_value + 0))
                fi
        done
        #If existing_value is equal to 0, this means the new host is not present, and shall thereby be added to the NMS.
        if [ $existing_value -eq 0 ];
        then
                #Add host to op5 via op5 monitor API and saves the $hostname in a temp-file (Why?).
                php /opt/monitor/op5/nacoma/api/monitor.php -t host -o host_name=$hostname -o address=$hostname -o template=default-host-template -u monitor
                echo $hostname >> /tmp/hostnames
                #Set save equals to one, which will indicate that changes has occured and thereby the Nacoma should save the configuration.
                save=1
        #If existing value is greather then 0 it means that the host is already present.
        elif [ $existing_value -gt 0 ];
        then
                echo "the node $hostname does already exist."
        else
                echo "Unknown error."
        fi
done
if  [ $save -eq 1 ];
then
        #Saving the configuration done by the op5 monitor API, which will then be presented on the WEB GUI.
        php /opt/monitor/op5/nacoma/api/monitor.php -a save_config -u monitor
else
        echo "Nothing to save. All hosts are present."
        exit 0;
fi
```

G - Add services linux

```bash
#!/bin/bash
#Script to add services for linux systems to the op5 monitor GUI.

#Read machines (hostnames) from file (machines)

machines=`cat /mnt/upload/hosts_op5.txt`

#For each $machine in $machines..
for machine in $machines
do
    #Read checks (full names for the checks)
    #example from /mnt/checks: command[users]=/opt/plugins/check_users -w 5 -c 10
    while read check
    do
        #Cutting of to the command_args, for example -w 5 -c 10
        command_args=$(echo $check | awk -F] '{print $1}'|awk -F[ '{print $2}')

        #Cutting of to the description of the service, for example Check Users.
        desc=$(echo "$command_args" | sed 's/_/\ /g')
        #Execute the op5 monitor API command to add new services, and using the variable for the hostname($machine), the service description ($desc), and the check_command_args ($command_args).
        try_check=`php /opt/monitor/op5/nacoma/api/monitor.php -t service -a show_object -n "$machine;$desc" -u monitor | grep check_command=`
        if [ ! -z "$try_check" ];
        then
            echo "The service $command_args is already present on node $machine."
        else
            echo "Adding $command_args to $machine."
            php /opt/monitor/op5/nacoma/api/monitor.php -t service -o template=default-service -o host_name="$machine" -o service_description="$desc" -o check_command=check_nrpe -o check_command_args="$command_args" -u monitor
        fi
    done < /mnt/upload/checks.txt

    #Add check-ping because it is not provided by NRPE. Checks if it already present before.
    check_ping=`php /opt/monitor/op5/nacoma/api/monitor.php -t service -a show_object -n "$machine;PING" -u monitor | grep check_command=`
    if [ ! -z "$check_ping" ];
    then
        echo "The service check_ping is already present on $machine."
    else
        echo "Adding check_ping to $machine."
        php /opt/monitor/op5/nacoma/api/monitor.php -t service -o template=default-service -o host_name="$machine" -o service_description="$desc" -o check_command=check_ping -o check_command_args=100,20%\!500,60% -u monitor
    fi

    #Add check-ssh-server because it is not provided by NRPE. Checks if it already present before.
    check_ssh=`php /opt/monitor/op5/nacoma/api/monitor.php -t service -a show_object -n "$machine;SSH Server" -u monitor | grep check_command=`
    if [ ! -z "$check_ssh" ];
    then
        echo "The service check_ssh is already present on $machine."
    else
        echo "Addinge check_ssh to $machine."
```

```
                php /opt/monitor/op5/nacoma/api/monitor.php -t service -
o template=default-service -o host_name="$machine" -
o service_description="SSH Server" -o check_command=check_ssh -
o check_command_args=5 -u monitor
        fi
done
#Saving the configuration done by the API, which will then be presented on the op5 m
onitor WEB GUI.
php /opt/monitor/op5/nacoma/api/monitor.php -a save_config -u monitor
```

H - Add nodes puppet

```
#!/bin/bash
#
#Add nodes to puppet configuration file site.pp.
#Example output in /etc/puppet/manifests/site.pp
#node 'examplecomputer.example.com' {include example_module}
#
#

#Read all nodes defined in /mnt/upload/hosts_op5.txt.
while read node
do
#Syntaxing the node definition correct with the module nrpe.
node_full="node '$node' { include nrpe }"
#Variable to check if configuration has occured.
exist_value=0
        #Read all existing node defitions in site.pp
        while read existing_node
        do
                #If the existing node is same as the new one in node_full..
                if [ "$existing_node" == "$node_full" ]
                then
                        #.. add + 1 to exist_value.
                        exist_value=$(($exist_value + 1 ))
                else
                        #If it is not, add nothing.
                        exist_value=$(($exist_value + 0 ))
                fi
        done < /etc/puppet/manifests/site.pp

#If exist_value is greather then 0 it means that the node is already present.
if [ $exist_value -gt 0 ];
then
        echo "Node $node already exist."
#Or if exist_value is equal to 0 it means that the node is not present and should then be added to site.pp.
elif [ $exist_value -eq 0 ];
then
        echo "Node $node added."
        echo "$node_full" >> /etc/puppet/manifests/site.pp
fi

done < /mnt/upload/hosts_op5.txt
```

I - Remove nodes puppet

```bash
#!/bin/bash
#
#Remove node-def. from /etc/puppet/manifests/site.pp, gained from /mnt/done.txt

#Checks how many lines there are in hosts_op5.txt. The number is equal to the number hostnames defined in hosts_op5.txt.
counter=$(wc -l < /mnt/upload/hosts_op5.txt )
#Variable to know if changes have occurred.
exist_value=0
#Read every nodes in hosts_op5.txt
while read uploaded_node
do
        #And read all nodes in done.txt
        while read done_node
        do
                #if the node in done.txt is equal to a node in hosts_op5.txt..
                if [ "$done_node" == "$uploaded_node" ];
                then
                        #.. add  + 1 to the exist_value.
                        exist_value=$(($exist_value + 1 ))
                else
                        #Otherwise add 0.
                        exist_value=$(($exist_value + 0 ))
                fi

        done < /mnt/done.txt

done < /mnt/upload/hosts_op5.txt
#If exist_value is equal to the counter.. (In other words: if the number of matches in done.txt compared to hosts_op5.txt is equal to the number of lines in hosts_op5.txt, do the following:)
#This means that done.txt contains all hostnames in hosts_op5.txt, and thereby are all nodes done with the configurations.
if [ $exist_value -eq $counter ];
then

        #If the archive-directory exist, move the file checks.txt and hosts_op5.txt to that directory.
        if [ -d /mnt/archived_services_and_hosts ];
        then
                mv /mnt/upload/hosts_op5.txt /mnt/archived_services_and_hosts/hosts_op5_$(date +%F)_$(date +%T)
                mv /mnt/upload/checks.txt /mnt/archived_services_and_hosts/checks_$(date +%F)_$(date +%T)
        else
                #If the directory does not exist, create it and then move the files.

                mkdir -p /mnt/archived_services_and_hosts/
                mv /mnt/upload/hosts_op5.txt /mnt/archived_services_and_hosts/hosts_op5_$(date +%F)_$(date +%T)
                mv /mnt/upload/checks.txt /mnt/archived_services_and_hosts/checks_$(date +%F)_$(date +%T)
        fi
        # For every node in done.txt, remove it from site.pp and then done.txt.
        while read node
        do
                node_full="node '$node' { include nrpe }"
                sed -i "/$node_full/d" /etc/puppet/manifests/site.pp
                sed -i "/$node/d" /mnt/done.txt
        done < /mnt/done.txt
        #Deletes the uploaded script, if it is uploaded.
        script_name=`ls /mnt/scripts/`
        if [ -z "$script_name" ];
        then
                echo "No script seems to be uploaded. Nothing to delete."
        else
                rm -f "/mnt/scripts/$script_name"
        fi
#Else, something is wrong.
else
        echo "The file /mnt/done.txt does not exist, which it should because this script is executed by its existence. It can also mean that the files are different, please wait 2-4 minutes."
fi
```

J - Puppet code for runinterval and usecacheonfailure

```
node default {

        file_line {'runinterval':
                ensure  => present,
                path    => '/etc/puppet/puppet.conf',
                line    => 'runinterval=120',
                before  => File_line['cache']
        }


        file_line {'cache':
                ensure  => present,
                path    => '/etc/puppet/puppet.conf',
                line    => 'usecacheonfailure=false'
        }

        exec {'service puppet restart':
                path    => ['/sbin', '/bin','/usr/sbin', '/usr/bin'],
                subscribe       => File_line['cache'],
                refreshonly     => true
        }
}
```