UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Mihkel Hani

Automated Grading System: The DevOps course Use Case

Bachelor's Thesis (9 ECTS)

Supervisor(s): Chinmaya Kumar Dehury, PhD

Tartu 2024

Automated Grading System: The DevOps course Use Case

Abstract:
Automated grading systems (henceforth referred to as AGS), have the goal of automating repetitive tasks, like homework grading. Using an AGS helps save time and minimise human error in the grading process.
As of the current moment (May 2024), grading in the course in question is being done manually by analysing and giving feedback to screenshots, texts, program snippets and IT systems. This creates a problem, where grading each homework manually consumes a lot of time and is prone to human error. The goal of this thesis is to create an automated grading system that supports creating new automated tests for the University of Tartu's course "DevOps: Automating Software Delivery and Operations" (LTAT.06.015), where homework is usually presented in the form of information technology (IT) systems. The job of the automated grading system is to use these tests to give immediate feedback on the students' work with minimal teacher intervention.
The created system can automatically grade all the homework in the course in question by using prewritten tests. The problem is solved since every test has to be written once, and thereafter, the system will use them to independently check if all the homework is solved correctly or not. Additionally, the system can automatically grade the students' solutions a lot faster than a human. To present the solution, an example: if originally it took around 5-10 minutes to grade the 1st homework by hand, then the newly created system can do it in seconds.

Keywords:
DevOps, Nagios Core, NRPE, software agent, automation, automated tests, automated grading system, cloud systems, monitoring, CI/CD

CERCS: P175 Informatics, system theory

Automatiseeritud hindamissüsteem: DevOpsi kursuse kasutusjuht

Lühikokkuvõte:
Automatiseeritud hindamissüsteemide (edaspidi AGS) eesmärk on automatsieerida korduvaid ülesandeid, nagu kodutööde hindamine. AGS-i aitab aega säästa ning vähendada inimviga hindamisprotsessis.
Hetkeseisuga (mai 2024), toimub hindamine manuaalselt kuvatõmmiste, tekstide, programmijuppide ja IT süsteemide analüüsimise ning tagasisidestamisena. Selle tagajärjeks on probleem, kus iga kodutöö käsitsi hindamine kulutab tohutult aega ning on toob sisse võimaluse inimveaks. Lõputöö eesmärgiks on luua automaathindamissüsteem, millele saab luua automaatteste, Tartu Ülikooli kursusele "DevOps: tarkvara tarnimise ja käituse automatiseerimine"(LTAT.06.015), kus kodutööd on üldiselt infotehnoloogia (IT) süsteemide kujul. Automaathindaja töö on kasutada neid automaatteste, et anda kodutöödele koheselt tagasisidet, õppejõu minimaalse sekkumisega.
Loodud süsteem suudab automaatselt hinnata kõnealusel oleva kursuse kodutöid kasutades ettekirjutatud teste. Probleem on lahendatud, kuna süsteemile tuleb ühe korra kirjutada testid ning seejärel kontrollib see iseseisvalt, kas kodutööd on korrektselt lahendatud või mitte. Lisaks sellele, et süsteem suudab automaatselt kontrollida tudengite lahendusi, teeb ta seda väga palju kiiremini kui inimene. Tulemuse näitena, kui algselt läks 1. kodutöö hindamisele õppejõul 5-10 minutit, siis loodud süsteem teeb seda sekunditega.

Võtmesõnad:
DevOps, Nagios Core, NRPE, tarkvara agent, automatiseerimine, automatiseeritud testid, automatiseeritud hindamissüsteem, automaathindajad, pilvesüsteemid, CI/CD

CERCS: P175 Informaatika, süsteemiteooria

Contents

1 Introduction

An information technology (henceforth referred to as IT) system is defined as a *set of one or more computers, associated software, peripherals, terminals, human operations, physical processes, information transfer means, that form an autonomous whole, capable of performing information processing and/or information transfer* [1]. An IT system, that operates autonomously is an automated system. Automated systems can be used for different goals, including grading. From now on, in the case of a system **A** that automatically grades some other system **B**, system **A** will be called a grading system and system **B** will be called a target system. In the context of this thesis, the target systems are mostly virtual machines, but also actively running systems like web services, that run in the cloud. The grading system automatically evaluates each target based on whether they are operating as intended.

Using an automated grading system (henceforth referred to as AGS) in the context of automating homework grading brings multiple benefits, such as:

- students getting fast feedback on their work and instantly knowing about their grades,
- teaching assistants saving time on grading repetitive homework,
- consistency in feedback, caused by minimising human error.

Although the benefits are great, there are also cons that cannot be overlooked. In this context, they are mainly:

- difficulties in actually setting the grading system up, due to specific needs for different tasks and therefore a lack of generalised solutions,
- requiring a thorough knowledge of what is needed to be graded and how, since ignorance can end up with not only a broken grading system but also a broken target system.

Although the problems that can be caused are significant, when recognising them and working to avoid the issues, then they are easily overshadowed by the pros.

This thesis is about implementing a grading system into the University of Tartu's course "DevOps: Automating Software Delivery and Operations" (LTAT.06.016)[2] (henceforth referred to as the DevOps course). The DevOps course states that *The primary objective of this course is to provide the basic understanding of core principles, practices and tools of DevOps*[2].

DevOps is a *methodology which combines together software development and IT operations in order to shorten the development and operations lifecycle* [3]. Because time is limited, then for most, a course about saving time in the IT sector is highly

relevant and streamlining this experience makes it more worthwhile. The additional benefit to this is that the course becomes inherently more scalable as the grading system can grade hundreds of target systems in a fraction of the time it takes for a human.

This thesis has been divided into five main parts. Section 2 talks about the DevOps course in more detail, introduces the tools used for this project and looks over existing solutions. Section 3 starts with the requirements for the project and then addresses the project structure. Section 4 is about how to implement this system. Section 5 shows the results of testing the system in a realistic scenario. Section 6 talks about the limitations and use cases for this system, what it can do and what it cannot. Last but not least, Section 7, the conclusion, discusses what was done in this thesis, the final result and future possibilities. Finally, in the appendix, there is information about how to access the source code, my acknowledgements and the licence.

I hereby confirm that I have created this thesis without any assistance from generative AI tools.

2 Background

The goal of this section is to give the reader some knowledge of the systems, platforms and tools related to this thesis.

2.1 The DevOps Course

In addition to the general goal of the DevOps course, it is beneficial to have some understanding of what the homework is and how they are presented and submitted.

2.1.1 Activities

In the DevOps course, students study about different topics [2]:

- The DevOps lifecycle,
- Cloud Computing + Containerization,
- Version Control Systems,
- Container Orchestration Systems,
- Microservice-based Application Development,
- Automation,
- Continuous Integration, Delivery and Deployment (CI/CD),
- Continuous Testing,
- Monitoring,
- Application Deployment Modelling,
- DataOps,
- Serverless Computing.

Many of these, if not all, are learned through using cloud services (like Microsoft Azure and Amazon Web Services) and virtual machines (henceforth referred to as VMs). The Estonian Scientific Computing Infrastructure (short form ETAIS) provides cloud services for the University of Tartu [4].

In the context of the DevOps course, the usage of ETAIS usually provides multiple benefits for the teaching assistants and the students:

- for homework, students get an isolated private environment in the form of a project,
- in each project, the corresponding student can order resources from ETAIS in the form of VMs and other cloud service-related resources,
- students are not limited by their own machine's capabilities and need minimal resources for solving the homework,
- students also gain experience in cloud services,
- teachers and teaching assistants gain the same benefits as the students,
- additionally, teachers and teaching assistants still have access to the students' projects, so that they can grade the homework tasks better,
- all the resources can run independently of their owners' machines.

Many of the benefits pointed out, also support integrating a grading system into the DevOps course.

2.1.2 Estonian Scientific Computing Infrastructure

ETAIS itself is a national infrastructure used for research that requires computing. It is managed by the University of Tartu, Taltech, the National Institute of Chemical Physics and Biophysics and the Education and Youth Board [5]. Because so many institutes manage ETAIS, it additionally provides excellent computing resources for low prices [6] and low latency within Estonia, since the server-client distances are small [7].

2.2 Tools

Choosing a tool for an IT project is made difficult due to the broad choice of tools. In this work, the focus is on automation and, therefore, automation tools. Through a careful elimination process, the main tool chosen for this, was Nagios. This section is divided into two parts, first briefly about the automation tools considered and second about the chosen automation tool, Nagios.

2.2.1 Considered automation tools

To choose a proper tool, many should be taken into comparison. The automation tools considered and why they were left out were [8]:

1. Ansible - Although powerful, it lacks built-in features, like a user interface (henceforth referred to as UI) and is slow since it does not use agents.

2. Puppet - Used for automating the configuration of servers, which is not in the scope of the current project.

3. Chef - May not work on all cloud providers and is mainly used for deploying, not monitoring.

4. Jenkins - Is not built for monitoring either.

5. Codeship - Although made for testing and deployment, it is built to check code that is merged into version control systems (henceforth referred to as VCS), which undermines the goal of monitoring the resources on ETAIS.

6. ELK - Resource-intensive and has a steep learning curve.

7. Terraform - Used to manage infrastructure, which is not necessary in the current context.

Having pointed out the weaknesses of each tool, it is still necessary to explain how Nagios is better than the other tools.

Firstly, Nagios is already built with the goal of monitoring systems [9]. Secondly, it offers both agent-based and agentless support, so if one becomes needed, the whole project does not need to be redone on a new tool [8]. Finally, the main giveaways are UI (refer to Figure 1) on the open source version, high customisability and maturity as a tool [8].

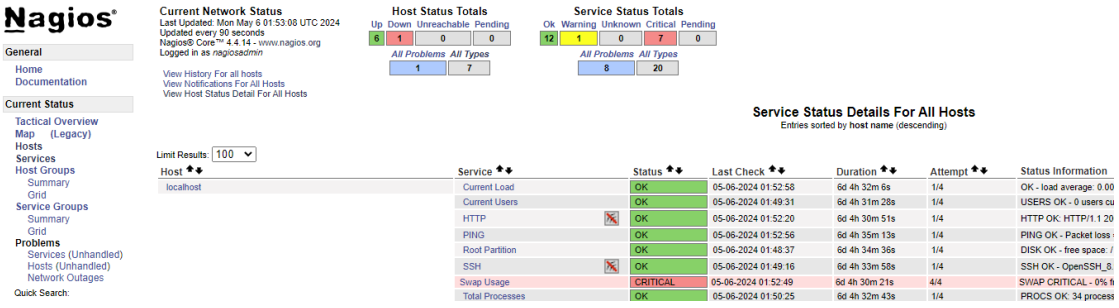


Figure 1. Screenshot of Nagios Core's UI

2.2.2 Nagios

Nagios is a monitoring system that allows different tools and plugins, mainly agent-based plugins, making it flexible for varying topics [10]. The mainly used plugin is called

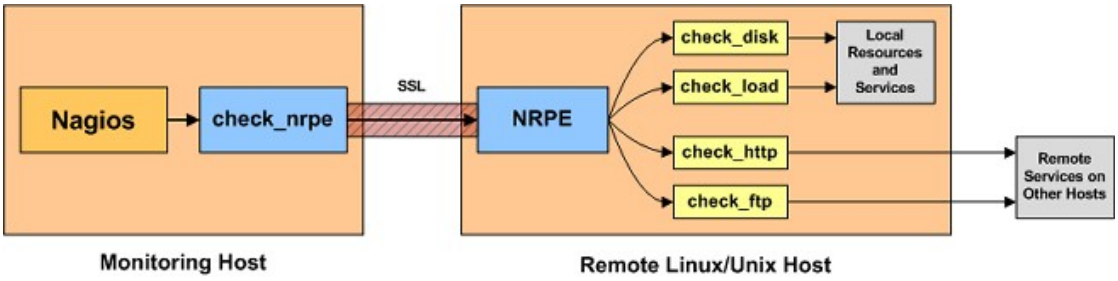


Figure 2. Nagios Remote Plugin Executor Design Overview [11]

Nagios Remote Plugin Executor (henceforth referred to as NRPE), which allows an agent-based architecture.

The monitored VMs will have an NRPE daemon (agent) running. The monitoring machine (Nagios Core) can contact the agents using the *check_nrpe* plugin, which can ask the daemon to execute a command (refer to Figure 2). [11]

2.2.3 Other tools

Additionally, there are tools used to support the system built on Nagios. These tools are:

- **GitHub**, which is a version control system (henceforth referred to as VCS) [12], is used in this thesis to store the installation scripts and keep the configurations up to date.
- **CronJob**, which is used for completing scheduled tasks repeatedly [13], is used in this thesis to add new hosts to the Nagios configurations automatically.
- **Bash** is the Linux scripting language, used in this thesis for scripting.
- **Python** is a programming language that is used in this thesis for a script that checks and creates configurations for hosts for Nagios configurations.

2.3 Existing solutions

As stated in Section 2.1.1, the lack of generalised solutions and finding related literature is a big difficulty for these types of projects. The few solutions that there are, do not provide code nor a public working solution [14, 15], and therefore cannot be used nor compared with the working solution of this project.

3 Proposed Automated Grading System

In this section, the project will be thoroughly examined.
Firstly, the requirements will be stated. Secondly, the system structure and its components are explained. Thirdly, how to implement the project. Last but not least, the testing section is a proof of concept of the system working in a real scenario. And lastly, the limitations of the system.

3.1 Requirements

For a functioning and useable system, some requirements need to be established. In the context of the DevOps course homework grading system, these are the following:

1. The grading and target systems should be easy to install.
2. The grading and target systems should have documentation on:
   (a) Installing the system
   (b) How the components of the system work
   (c) How to configure the system
   (d) How to add new tests for homework to the system
3. The grading system should have configurable tests for homework.
4. The grading system should work with many target systems.
5. On the ETAIS cloud service, the target systems should be automatically detected and added to the list of targets.
6. The grading system should be able to operate autonomously most of the time without the interference of a third party.
7. For more detailed homework tests, the grading system should be able to execute commands and retrieve their output on the target system.

3.2 Structure

The system can be divided into two main parts: the grading system (VM running Nagios Core) and the target systems (VMs running NRPE agents). In general, the working of the system is like the following: the Nagios Core looks at what machines need to be checked, sends check requests, and if the machines have an NRPE agent, they respond with a message and status (refer to Figure 3).

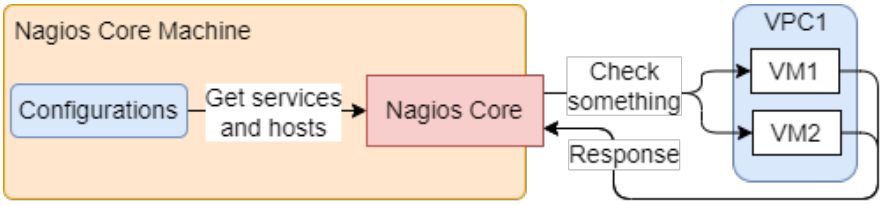


Figure 3. The general working of the system.

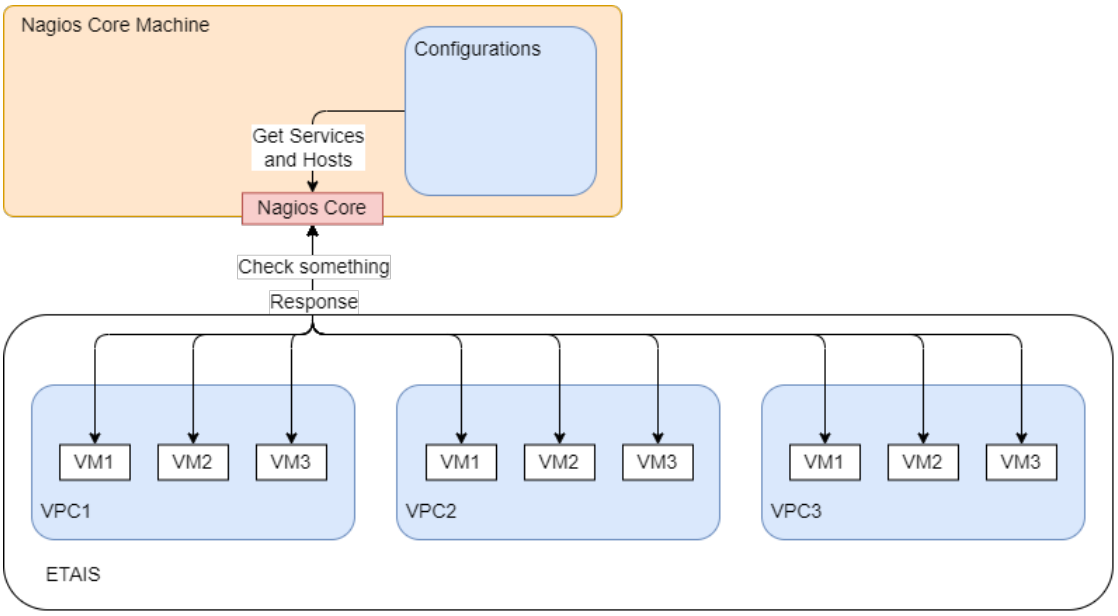


Figure 4. The general working of the system with ETAIS.

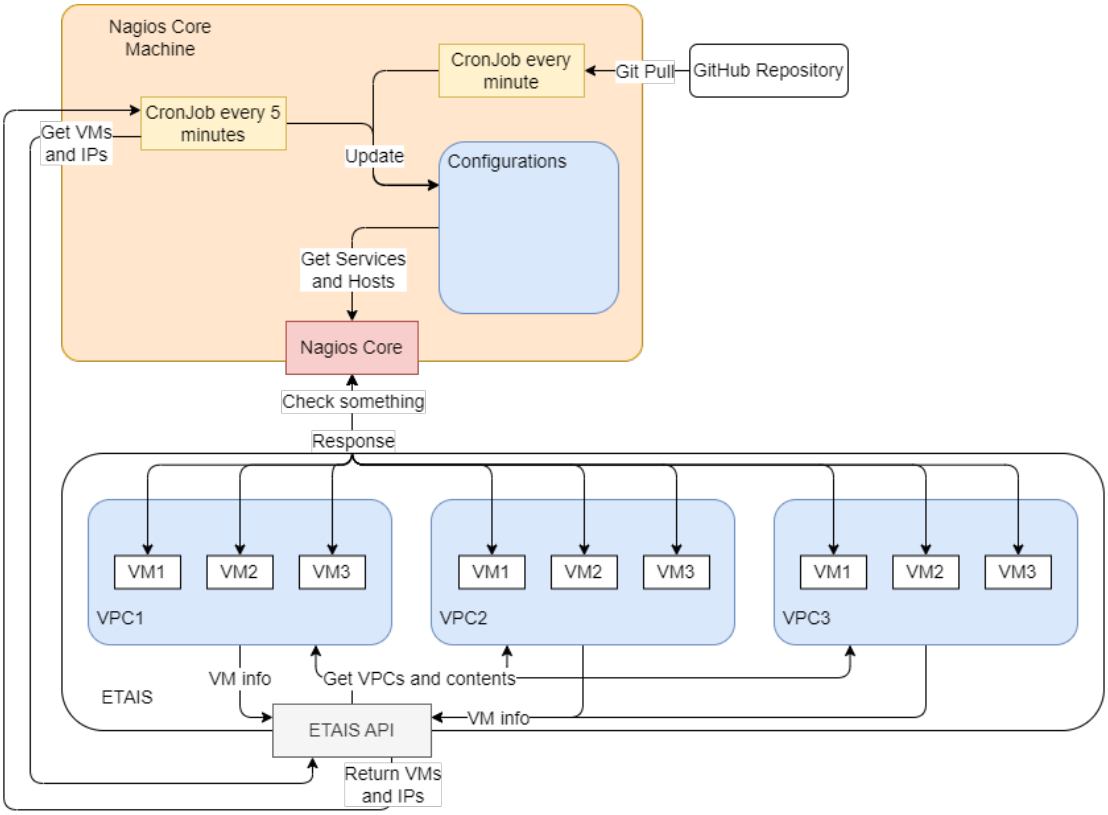


Figure 5. The general working of the entire system

The complexity can be further increased, by bringing the ETAIS platform into the picture (refer to Figure 4). And further enhanced by using a CronJob that checks for new hosts in the ETAIS system and a CronJob that checks for updates in GitHub. (refer to Figure 5).

3.2.1 The grading system

The grading system is the machine that hosts the UI, stores and updates configurations and runs the checks for the target systems.

The grading system itself is divided into three parts: the configurations, the configurations updater and the Nagios Core service (Nagios Core service includes the UI as well).

The configurations The configurations are files with a certain style defined by the Nagios Documentation. Superficially, the used configuration settings are:

- Configuration file and directory paths, which define additional configuration files'

paths.

- Host definitions, which define hosts (the targets' IP addresses) that Nagios Core will try to check.
- Hostgroup definitions, which define groups for hosts. These can be used to group hosts, by adding the field "hostgroup" to a host definition. This is useful for the next definitions.
- Service definitions, which define the tests that are run. These can be applied to an individual host or a hostgroup. By applying it to a hostgroup, the service will be run on all hosts (refer to Figure 6). Service definitions also have a name, description and, most importantly, the command to be checked fields.
- Command definitions are used to define commands so that the Nagios Core can run them. A command is defined by a name and the command-line field, which shows what command-line command to execute when called.

The command definitions also take parameters and can be given in service definitions (more on this later).

DevOpscourse-internal-NagiosTest	base-service	OK	05-06-2024 04:48:58	6d 7h 15m 22s	1/3	PING OK - Packet loss = 0%, RTA = 1.22 ms
	hw1 test1 custom script	CRITICAL	05-06-2024 04:46:06	4d 11h 28m 14s	3/3	(No output on stdout) stderr: connect to address 172.17.88.91 port 5666: Connection refused
DevOpscourse-internal-TestStudent1	base-service	CRITICAL	05-06-2024 04:45:22	6d 7h 18m 58s	3/3	CRITICAL - Host Unreachable (172.17.91.187)
	hw1 test1 custom script	CRITICAL	05-06-2024 04:46:54	4d 11h 37m 26s	3/3	(No output on stdout) stderr: connect to address 172.17.91.187 port 5666: No route to host
DevOpscourse-internal-TestStudent2	base-service	OK	05-06-2024 04:47:33	5d 11h 49m 37s	1/3	PING OK - Packet loss = 0%, RTA = 0.90 ms
	hw1 test1 custom script	WARNING	05-06-2024 04:47:29	4d 5h 6m 52s	3/3	Warning - maybe works?

Figure 6. Example of the same script being run on multiple machines.

The configurations updater The configurations updater is a system built using CronJob, and it also has two parts to it:

1. The necessary host configurations updater (henceforth host updater).
2. The optional CI/CD system.

The host updater runs every 5 minutes (unless configured otherwise), where it runs a Bash script *run_get_students.sh*. The script has 3 tasks:

1. Run the Python script *check_VMs_status.py*. The script checks for projects in the ETAIS system and removes the ones that are not wanted (refer to Listing 1). Then, the script will check the VMs for each project and get their names and IP addresses. The final name, which will be shown in the UI, is made by combining the name of the project and VM (Example: Project-VM_Name). Finally, using each name and IP address pair, the host definitions are created and put into a file.

2. Run the *update_config_files.sh* script, which sends all the configuration files in the current directory, to the "*/usr/local/nagios/etc/objects*" directory. This is necessary since Nagios Core has trouble reading configuration files in other directories than starting with the path "*/usr/local/nagios*".

3. Finally, it will call the *systemctl reload-or-restart nagios.service* command, which restarts the Nagios service. If this is not done, the updates to the configuration files will not be enforced.

```
exclude_project_list = ["DevOps2021Fall", "DevOpscourse- internal"]
```

Listing 1. Example of excluded projects. In this context „DevOps2021Fall“ and „DevOpscourse-internal“ are excluded.

The optional CI/CD CronJob runs every minute (unless configured otherwise). Because it is more complex, it is advised to look at Listing 2, while reading the following text.

First of all, it runs the git pull command using the ubuntu user (default is sudo, which is escalated privileges, but GitHub strongly recommends against using sudo for git [16]). Then using "*&& echo*" right after git pull, forces it to wait until git pull has returned an output. This is necessary since the output will be passed to the *ci-cd.sh* script, which checks if any changes were made by the *git pull* action. If not, then the *ci-cd.sh* script does nothing, otherwise the *ci-cd.sh* script will run the *update_config_files.sh* script and restart the Nagios service.

```
*/1 * * * * echo "$(su -s /bin/sh ubuntu -c 'cd ~/LabEvalAutomation
    && /usr/bin/git pull && echo')" | /home/ubuntu/LabEvaluation/ci-cd
    .sh
```

Listing 2. The CI/CD CronJob

The advantage of using the optional CI/CD system comes out when developing. While developing, the automatic updating allows to avoid unnecessary and repetitive commands, which in turn speeds up the developing process.

The Nagios Core Service The system has been set up in such a manner that there are some base definitions. This helps avoid unnecessary repetition and keeps the code cleaner thanks to the fact that Nagios definitions support inheritance. For easier understanding

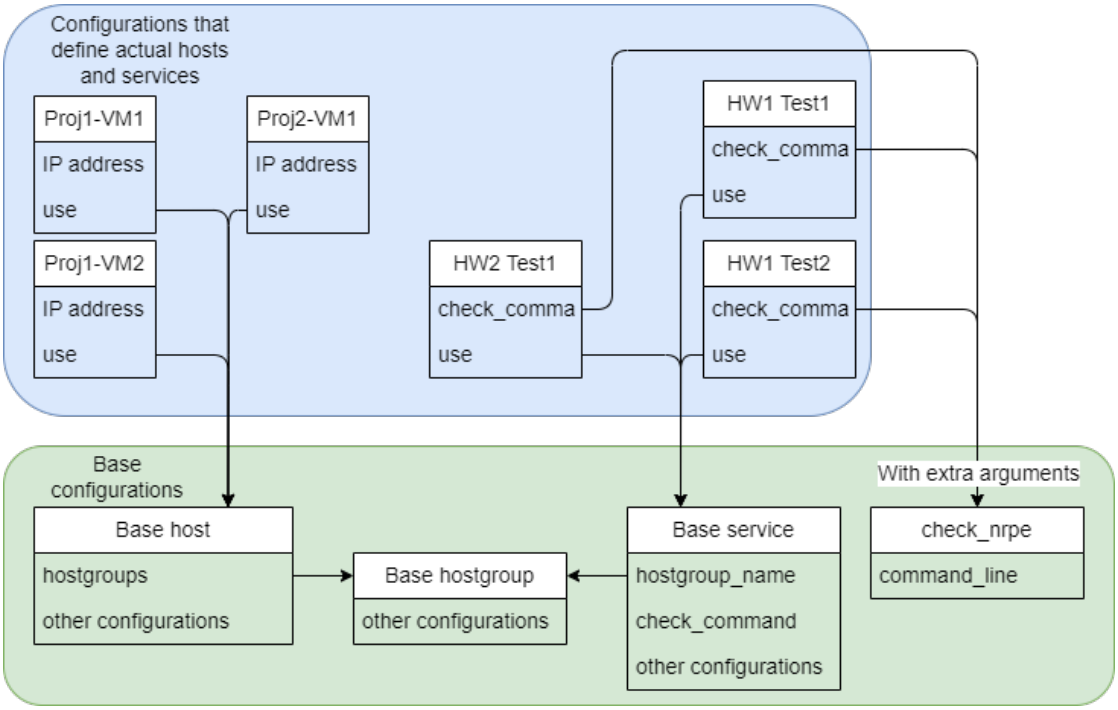


Figure 7. Structure of configurations

of the following, refer to Figure 7. There are 3 “Base” definitions: Base Host, Base Hostgroup and Base Service. Also there is one custom (meaning not built-in) command definition, which will be talked about later, after the NRPE agents have been explained.

The Base Host has a field hostgroups and contains one: Base Hostgroup. Additionally, there are multiple fields that define how often checks are being done on the host. This allows all hosts that inherit the Base Host, to use the same configurations.

Similarly, the Base Service has a field hostgroup, which contains Base Hostgroup and allows for each service that inherits it to use the group by default.

The effect of both of these configuration choices is that all services which inherit the Base Service apply to all hosts that inherit the Base Host.

With this solution, the Nagios Core service will check its configurations and send the commands defined in service definitions to the IP addresses defined by host definitions.

3.2.2 The targets

The NRPE agents are configured a lot simpler. They have the *5666/TCP* port allocated to the NRPE service. If an IP address that has been marked into the trusted configurations, sends the *check_nrpe* command, then the NRPE agent will try to execute it according to its configurations and respond accordingly (refer to Figure 8). Step-by-step, Nagios

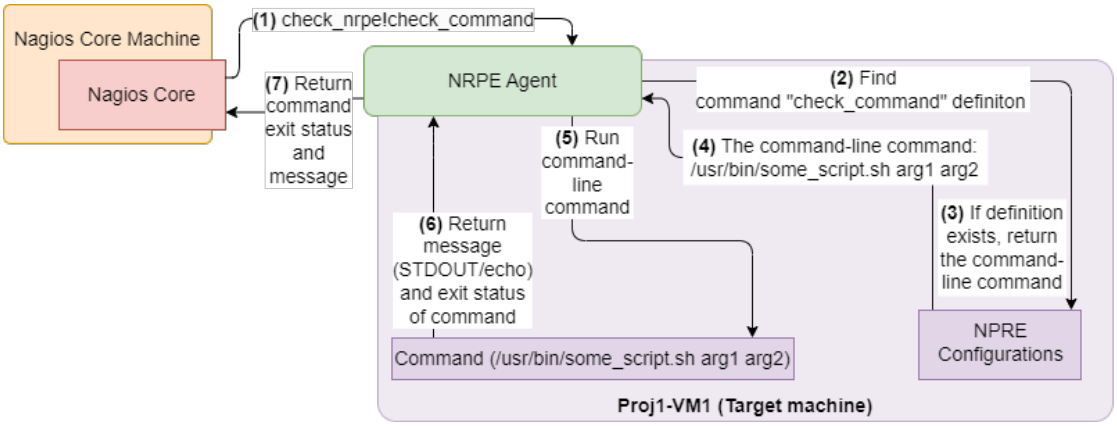


Figure 8. NRPE agent workflow

Core executes the *check_nrpe* plugin with the following arguments: hostaddress, port, command and arguments.

If there is an NRPE agent listening on the hostaddress (IP address defined in the host definition) and port (5666), which accepts this Nagios Core IP address, then it will take the arguments: command and arguments. One of the arguments sent with *check_nrpe* is the command that needs to be executed by the NRPE agent. Upon finding the command definition matching this argument, the agent will take and execute the command-line value assigned to it. Then, it will respond with the message and exit status of the command.

The message is any string in the *STDOUT* of the script, which is in simple terms, the value printed to the console.
The exit status is either:

- 0 – OK (meaning everything is fine),
- 1 – WARNING (meaning something may cause issues),
- 2 – CRITICAL (meaning something is wrong).

It is also a good practice to format the message using the matching keyword:

- 0 => OK – Everything is fine
- 1 => WARNING – Something may cause issues
- 2 => CRITICAL – Something is wrong

Note: The messages at the end are just placeholder messages and could be replaced with anything. Also, adding the keyword to the start is unnecessary, but recommended.

4 Implementation

This subsection shows how to implement this system in the use case of the DevOps course.

4.1 Installation

The installation process for Nagios Core and the NRPE agent might seem complex for beginners. The reason is that by googling, one can find multiple different ways to install Nagios Core [17, 18, 19] just from their own websites. A similar issue arises with installing the NRPE plugin [20, 21, 22]. Additionally, when installing, it might not be clear which steps to skip and which to complete [19]. In the case of the DevOps course, the correct solution for this problem was devised from extensive research of Nagios and NRPE by reading about and trying the different solutions. The result is guides and scripts that help with the installation and configuration of the grading and target systems, leaving only a few extra steps to the configuring of the system.

4.2 Installation of the grading system

The first part of the installation process is to set up the grading system. For that, in the context of the DevOps course, a VM running Linux Ubuntu version 22.04 with 2 CPU cores and 4 GB of RAM in the ETAIS cloud is used. After creating the VM, the next step is to connect to it.

Inside the created VM, an SSH (Secure Shell) private-public key pair is created. The public key is put into the GitHub keys of the user who has access to the grading system repository. The next step is to clone the repository and enter it. After that, the user should run the script *install_service.sh* using elevated privileges by running the command in Listing 3. Running the script will prompt the user to enter their ETAIS API token and the admin user password they wish to use. When prompted, the user should give these values. The script will update and install all the necessary Linux packages, then download the Nagios Core version 4.4.14 and install it. Additionally, the script creates a passwordless guest account, which can view the results of the tests when connecting to the UI.

```
sudo bash install_service.sh
```

Listing 3. Running the install_service.sh script

After this, the user should restart the Apache2 service and start the Nagios service using the two commands in Listing 4.

```
sudo systemctl restart apache2.service
sudo systemctl start nagios.service
```

Listing 4. Commands for restarting the Apache2 service and starting the Nagios service

The next step is to install the plugins and restart the Nagios service, using the commands in Listing 5. The first command runs the *install_plugins.sh* script, which updates-downloads-installs the necessary packages and files for the plugins.

```
sudo bash install_plugins.sh
sudo systemctl restart nagios.service
```

Listing 5. Commands to install the plugins and restart the Nagios service

To give Nagios Core the ability to communicate with NRPE agents, it needs the NRPE plugin. This can be installed with the command in Listing 6. It is similar to the script run in Listing 5, but downloads a different package and reloads the Nagios service.

```
sudo bash install_nrpe.sh
```

Listing 6. Command to install the NRPE plugin on Nagios Core

Last but not least, the configuration files should be initialized, so Nagios Core knows where to look for new configuration files. The command for that is provided in Listing 7.

```
sudo bash add_configs.sh
```

Listing 7. Commands to initialise Nagios configurations

Finally, for Nagios Core, the CronJob, which automatically updates the list of students, should be made. This is done by giving execute permissions on the *run_get_students.sh* script and then running a special command which appends it to the elevated privileges' CronJobs list. These commands are provided in Listing 8.

```
sudo chmod u+x run_get_students.sh
{ sudo crontab -l; echo "*/5 * * * * cd /home/ubuntu/
    LabEvalAutomation && ./run_get_students.sh"; } | sudo crontab -
```

Listing 8. Commands to create a CronJob which updates the students list every 5 minutes

Optionally, a CI/CD system can be implemented using CronJob. This CronJob will run the *git pull* command and pass the result to another script, every minute. The script will evaluate if the configurations need to be updated, and the Nagios service will be restarted based on that. Unlike implementing the students list CronJob, this cannot be done with only running 2 commands due to the way it is built. Instead, it is needed to open the CronJobs list in edit mode and copy the CronJob directly into there on a new line. The commands for giving execute permissions to the *ci-cd.sh* script and opening the CronJobs list can be found in Listing 9 and the CronJob, which needs to be added to the list, in Listing 10.

```
sudo chmod u+x ci-cd.sh
sudo crontab -e
```

Listing 9. Commands to add execute permissions to the ci_cd.sh script and open the CronJobs list

```
*/1 * * * * echo "$(su -s /bin/sh ubuntu -c 'cd ~/LabEvalAutomation
    && /usr/bin/git pull && echo')" | /home/ubuntu/LabEvaluation/ci-cd
    .sh
```

Listing 10. CronJob which needs to be added to the CronJobs list

4.3 Installation of a target system

The other side of the system, the target systems, will be created by the students in the DevOps course use case. For that, in the context of the DevOps course, a VM running Linux Ubuntu version 22.04 in the ETAIS cloud is used. After creating the VM, the next step is to connect to it.

The steps a student needs to do are the following and the corresponding commands can be found in Listing 11:

1. Download the installation script from the public GitHub repository, which the teacher provides.
2. Add execute permissions for the script.
3. Run the script with an additional parameter, which is the IP address of the grading system (in the example case, 172.17.89.137).
4. Finally, add the Nagios user to the sudoers list (the lines that need to be added to the sudoers list, are provided in Listing 12).

```
wget --no-check-certificate -O install.sh https://raw.
    githubusercontent.com/DevOps-TextBook/LabEvalAutomation-public/
    main/install.sh
sudo chmod u+x install.sh
sudo ./install.sh 172.17.89.137
sudo visudo
```

Listing 11. The commands a student needs to run to install the NRPE agent to their system

```
Defaults:nagios !requiretty
nagios  ALL=(ALL) NOPASSWD: ALL
```

Listing 12. The 2 lines needed to be added in the sudoers file which is opened when running sudo visudo

Superficially, the installation script updates-downloads-installs the necessary packages and files for the NRPE agent, then opens the *5666/TCP* port for communication with the Nagios Core and allows the given IP address to ask about the status of this machine. Additionally, the installation script:

1. adds a command definition to the NRPE configurations file, which runs the commands given to it through the arguments,
2. adds a line which overrides the default illegal characters in command arguments, into the NRPE configurations file,
3. changes the value of *allow_bash_command_substitution* to true in the NRPE config files, which allows for more complex instructions to be sent to the NRPE agent,
4. adds the Nagios user to the Ubuntu group so it can access files and directories without having to give access to each file and directory every time they are created,
5. restarts the NRPE service.

The many steps that have to be done in the usual installation are packaged in a single script for a streamlined experience when doing homework in the DevOps course.

4.4 Configuring

For Nagios Core and NRPE agents to communicate, some configurations have to be set. In this thesis, most of this is done automatically by the installation, the CronJobs and the *check_VMs_status.py* script. Nonetheless, the homework grading tests have to be manually created, and in the case of wanting to check different things for different types of hostgroups, a new host, hostgroup and service need to be defined. The *check_VMs_status.py* script already checks for 3 different types of host types (Listing 13). The way the script differentiates these is by the VM's name (if it contains the host type, then it is classified as it). In the case of a new hostgroup, the new name should be added to this list of host types.

```
host_types = ["master", "worker", "base"]
```

Listing 13. The line which contains different host types on row 83 of the check_VMs_status.py script

When creating hostgroups, services and hosts, the name should follow the convention on Listing 14.

```
name-host
name-hostgroup
name-service
```

Listing 14. The naming convention for new definitions

When creating a new command definition, it should have a name and the command it executes on the command-line. The command has to have an absolute path, and custom parameters need to be defined in the following manner: *$ARG1$ $ARG2$*, etc. All of these definitions should be added to the *homework_base.cfg* file.

As an example, it is necessary to check *custom_command.sh* on VMs that have the value *hosttype* in their names. For that, the configurations in Figure 9 are added to the *homework_base.cfg* file.

Then, to use the command, the service in Figure 10 can be added to the *homework* directory in the GitHub repository.

4.5 Services and commands

Nagios Core and NRPE agents have a special way of configuring their commands and services, which are used for setting up automatic tests. Due to the lack of explanation on how the arguments are passed from the service definition to the command definition to the NRPE agent and from there to the script that will be run, the process of figuring out a system takes longer than expected.

4.5.1 The issue

Inherently, Nagios Core and NRPE do not support storing scripts on Nagios Core and then activating them on NRPE. It has been made in a way, that Nagios Core can call a script on NRPE and then get the result. The naive approach would be to create scripts on the target machine, as initially expected from Nagios. In the context of DevOps homework automation, this has a major flaw: the student has *sudo* access to their machine (the target) and, therefore, can edit the scripts to give the "OK" result 100% of the time. Another idea could be setting up a repository that contains these scripts and, when needed, pulls them and runs them. This still has the flaw of needing a script that pulls the repository and responds with the script exit status and message. Creating a script file gives the student access to hardcoding an "OK" status for them. Compiling the files would not help either, since if the student gains knowledge on how the statuses are shown, they can make their own script, compile it and replace the existing script.

4.5.2 Approaching the solution on NRPE

Usually, Linux systems run their scripts using an interpreter called bash. In Linux, most commands/scripts, including bash, can be accessed and called no differently than any script. Additionally, running commands through bash is no different than running a bash script, because a bash script is just a set of instructions for the bash interpreter. Using the bash interpreter is the obvious choice for this task. That is because when calling bash, the flag "-c" allows passing the commands that need to be executed by bash into a single bash session without entering it, while still getting the result. With this in mind, it is possible to set the command definition in the target system like in Figure 11.

```
### Hosttype Host ###
define host {
        name     hosttype-host
        use      base-host
  hostgroups  hosttype-hostgroup
}
### Hosttype hostgroup ###
define hostgroup {
        hostgroup_name  hosttype-hostgroup
  alias Hosttype Hostgroup
}
### Hosttype Service ###
define service {
        name     hosttype-service
        use      generic-service
        hostgroup_name        hosttype-hostgroup
        service_description      hosttype-service
        check_command    check-host-alive
}

### Custom Command executed on NRPE agent ###
define command {
  command_name custom_command
  command_line /usr/local/nagios/libexec/check_nrpe -H $HOSTADDRESS$
      -p 5666 -c custom_command -a $ARG1$ $ARG2$
}

### OR Custom Command executed on the local machine ###
define command {
  command_name custom_command
  command_line /usr/local/custom_scripts/custom_command.sh $ARG1$
     $ARG2$
}
```

Figure 9. Example custom definitions added

```
define service {
        use      master-service
        service_description      HW1 Flask app
        check_command    custom_command!arg1!arg2
}
```

Figure 10. Examples of using the new commands

```
command[run_script]=/usr/bin/bash -c '$ARG1$'
```

Figure 11. Command definition used for NRPE

Some things to be taken into consideration using this approach are that:

- Each script needs to be written on one line, similar to when running a full command on one line in the shell, meaning that different commands need to be separated by a semicolon (command1; command2; command3; etc).
- The commands must be surrounded by single quotes ('command1; command2; command3; etc'), which is done automatically in the command definition (refer to Figure 11).

4.5.3 A special case

In the configuration files of NRPE, it is stated that uncommenting a line *nasty_metachars*=... (the three dots symbolize the nasty metacharacters), will override the default nasty metacharacters. Nasty metacharacters define what characters cannot be passed to the NRPE daemon. Although not mentioned directly, it can be assumed that they are the ones commented out in the configuration file (refer to Figure 12). These nasty metacharacters conflict with the command definition because it is necessary to pass semicolons and other symbols that bash uses through arguments. That is why, at the end of the *nrpe.cfg* file, which is used to define configurations in NRPE, there is a line like on Figure 13 (Instead of <insert_some_symbol> there is the symbol "~", which overleaf has difficulty rendering). Note: if this character becomes necessary, then it can be replaced with another character, that is not used. Using this solution, the NRPE agent can take input as a script with commands separated by semicolons and respond with its exit status and message.

```
# NASTY METACHARACTERS
# This option allows you to override the list of characters that
    cannot
# be passed to the NRPE daemon.

# nasty_metachars=|`&><'\\[]{};\r\n
```

Figure 12. Nasty metacharacters given by NRPE.

```
nasty_metachars=<insert_some_symbol>
```

Figure 13. Overwritten nasty metacharacters.

4.5.4 The solution on Nagios Core

The next part of this solution is configuring the Nagios Core to use the command. The command on Nagios Core is defined as *check_nrpe* and runs a command defined by the *command_line* field (refer to Figure 14). There are multiple aspects to this command, which are the following:

- */usr/local/nagios/libexec/check_nrpe* is the script/plugin that will execute using the given arguments. It is used for communicating with NRPE agents.
- *-H $HOSTADDRESS$* means that the host (*-H*) that will be contacted with, using the *check_nrpe* command, is *$HOSTADDRESS$*. The value *$HOSTADDRESS$* is automatically taken from the host definitions when Nagios Core performs checks on the NRPE agents.
- *-p 5666* stands for the port 5666, which will be used to contact NRPE agents (refer to Section 3.2.2).
- *-c run_script* means that the command executed on the host is *run_script*.
- *-a '$ARG1$'* means that the arguments are *'$ARG1$'* (there can be more arguments using *$ARG2$*, *$ARG3$*, ...). The value for *$ARG1$* comes from the service definition talked about right after this.

This command can be used in a service definition to be used for testing. Some notes about this approach:

- When making a new service, the command is defined as follows: *command_name!argument1!argument2!etc* (similarly to Figure 15).
- Since the script has to be sent as a string, it should be surrounded by single quotes. This has been done automatically in the command definition (refer to Figure 14) to avoid unnecessary repetition of writing the single quotes.
- Due to the necessity of single quotes, they cannot be used for making scripts, since *$ARG1$* will be replaced entirely by the argument, therefore, when using single quotes, the result will look something like the following: *'textbeforequote'textinquote'textafterquote'*; which is 2 strings *'textbeforequote'* and *'textafterquote'* separated by *textinquote*.

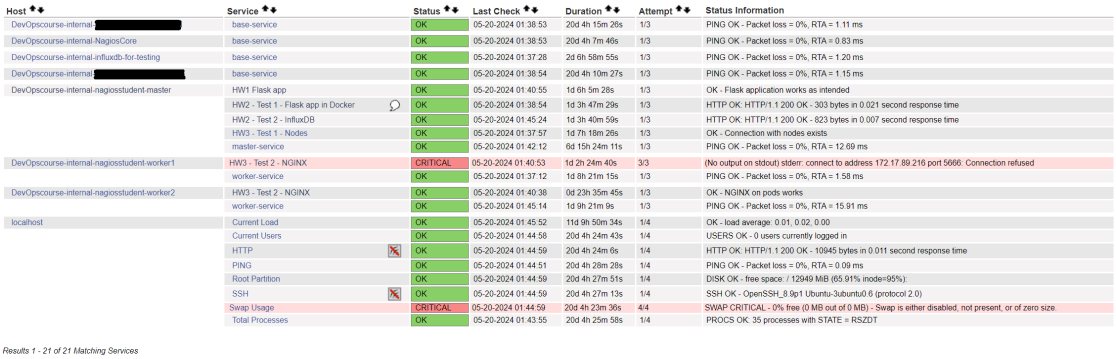
```
define command {
  command_name check_nrpe
  command_line /usr/local/nagios/libexec/check_nrpe -H $HOSTADDRESS$
      -p 5666 -c run_script -a '$ARG1$'
}
```

Figure 14. Command defintion used for Nagios Core

```
define service {
        use      base-service
        service_description      Example service
        check_command    check_nrpe!echo "Warning - some message"\;
            exit 1;
}
```

Figure 15. Example service definition

- Something that is not immediately apparent is that using a semicolon anywhere instantly ends the line. That means, when making scripts for testing purposes, the separating semicolon needs to be exited with an exit-character, such as in the Figure 15.



Host	Service	Status	Last Check	Duration	Attempt	Status Information
DevOpscourse-internal-[redacted]	base-service	OK	05-20-2024 01:38:53	20d 4h 15m 26s	1/3	PING OK - Packet loss = 0%, RTA = 1.11 ms
DevOpscourse-internal-NagiosCore	base-service	OK	05-20-2024 01:38:53	20d 4h 7m 46s	1/3	PING OK - Packet loss = 0%, RTA = 0.83 ms
DevOpscourse-internal-influxdb-for-testing	base-service	OK	05-20-2024 01:37:28	2d 6h 58m 55s	1/3	PING OK - Packet loss = 0%, RTA = 1.20 ms
DevOpscourse-internal-[redacted]	base-service	OK	05-20-2024 01:38:54	20d 4h 10m 27s	1/3	PING OK - Packet loss = 0%, RTA = 1.15 ms
DevOpscourse-internal-nagiosstudent-master	HW1 Flask app	OK	05-20-2024 01:40:55	1d 6h 5m 28s	1/3	OK - Flask application works as intended
	HW2 - Test 1 - Flask app in Docker	OK	05-20-2024 01:38:54	1d 3h 47m 29s	1/3	HTTP OK: HTTP/1.1 200 OK - 303 bytes in 0.021 second response time
	HW2 - Test 2 - InfluxDB	OK	05-20-2024 01:45:24	1d 3h 40m 59s	1/3	HTTP OK: HTTP/1.1 200 OK - 823 bytes in 0.007 second response time
	HW3 - Test 1 - Nodes	OK	05-20-2024 01:37:57	1d 7h 18m 26s	1/3	OK - Connection with nodes exists
	master-service	OK	05-20-2024 01:42:12	6d 15h 24m 11s	1/3	PING OK - Packet loss = 0%, RTA = 12.69 ms
DevOpscourse-internal-nagiosstudent-worker1	HW3 - Test 2 - NGINX	CRITICAL	05-20-2024 01:40:53	1d 2h 24m 40s	3/3	(No output on stdout) stderr: connect to address 172.17.89.216 port 5666: Connection refused
	worker-service	OK	05-20-2024 01:37:12	1d 8h 21m 15s	1/3	PING OK - Packet loss = 0%, RTA = 1.58 ms
DevOpscourse-internal-nagiosstudent-worker2	HW3 - Test 2 - NGINX	OK	05-20-2024 01:40:38	0d 23h 35m 45s	1/3	OK - NGINX on pods works
	worker-service	OK	05-20-2024 01:45:14	1d 9h 21m 9s	1/3	PING OK - Packet loss = 0%, RTA = 15.91 ms
localhost	Current Load	OK	05-20-2024 01:45:52	11d 9h 50m 34s	1/4	OK - load average: 0.01, 0.02, 0.00
	Current Users	OK	05-20-2024 01:44:58	20d 4h 24m 43s	1/4	USERS OK - 0 users currently logged in
	HTTP	OK	05-20-2024 01:44:59	20d 4h 24m 6s	1/4	HTTP OK: HTTP/1.1 200 OK - 10945 bytes in 0.011 second response time
	PING	OK	05-20-2024 01:44:51	20d 4h 28m 28s	1/4	PING OK - Packet loss = 0%, RTA = 0.09 ms
	Root Partition	OK	05-20-2024 01:44:59	20d 4h 27m 51s	1/4	DISK OK - free space: / 12949 MiB (65.91% inode=95%):
	SSH	OK	05-20-2024 01:44:59	20d 4h 27m 13s	1/4	SSH OK - OpenSSH_8.9p1 Ubuntu-3ubuntu0.6 (protocol 2.0)
	Swap Usage	CRITICAL	05-20-2024 01:44:59	20d 4h 23m 36s	4/4	SWAP CRITICAL - 0% free (0 MB out of 0 MB) - Swap is either disabled, not present, or of zero size
	Total Processes	OK	05-20-2024 01:43:55	20d 4h 25m 58s	1/4	PROCS OK: 35 processes with STATE = RSZDT

Results 1 - 21 of 21 Matching Services

Figure 16. Example of Nagios Core UI before creating a student machine

5 Testing

This subsection is about testing the system as a proof-of-concept using actual homework tasks. We have chosen a few homework tasks to check that the implementation is working.

5.1 Homework 1: Deploying a Flask web application

The first homework shows the ETAIS platform and how to create a VM there, with the goal of teaching how to deploy a simple Flask application in the cloud.

The reason this homework was chosen was because the result of it is a Python application which stops running after the student disconnects from the VM and when running, it occupies the active command-line interface. That poses a challenge, where the NRPE agent must have enough versatility and privileges to run the application and check its output at the same time. The solution uses a bash built-in command *tmux*, which allows multiplexing the terminal. It is used to run the Flask application in one terminal and then, in the other terminal, check if it is actually running. Finally, the results are reported back to the Nagios Core.

Initially, the student is presented with some instructions on how to connect to the ETAIS platform, create a new VM and connect to it. Before creating one, there are no configurations created for the VM in the Nagios Core. Therefore nothing will show up on the UI (refer to Figure 16).

When no errors occur, the system will take up to 5 minutes to find the student machine and update the configurations accordingly. The student machine will become detectable after it has reached the active status in ETAIS (refer to Figure 17).

After detecting the new VM, the Nagios Core UI will look like something in Figure 18.

As it can be seen, the services are still pending (gray), which means they have yet

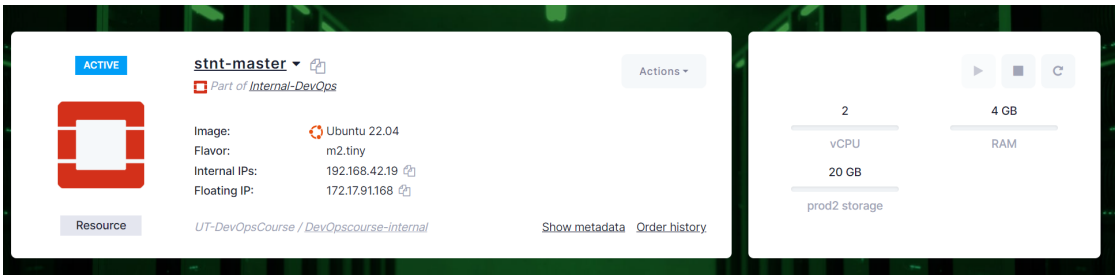


Figure 17. Example of student machine reaching active status in ETAIS

Host	Service	Status	Last Check	Duration	Attempt	Status Information
DevOpscourse-internal-[redacted]	base-service	OK	05-20-2024 01:48:53	20d 4h 24m 8s	1/3	PING OK - Packet loss = 0%, RTA = 1.34 ms
DevOpscourse-internal-NagiosCore	base-service	OK	05-20-2024 01:48:53	20d 4h 16m 28s	1/3	PING OK - Packet loss = 0%, RTA = 0.94 ms
DevOpscourse-internal-influxdb-for-testing	base-service	OK	05-20-2024 01:47:28	2d 7h 7m 37s	1/3	PING OK - Packet loss = 0%, RTA = 1.14 ms
DevOpscourse-internal-[redacted]	base-service	OK	05-20-2024 01:48:54	20d 4h 19m 9s	1/3	PING OK - Packet loss = 0%, RTA = 1.89 ms
DevOpscourse-internal-nagiosstudent-master	HW1 Flask app	OK	05-20-2024 01:50:55	1d 6h 14m 10s	1/3	OK - Flask application works as intended
	HW2 - Test 1 - Flask app in Docker	OK	05-20-2024 01:48:54	1d 3h 56m 11s	1/3	HTTP OK: HTTP/1.1 200 OK - 303 bytes in 0.021 second response time
	HW2 - Test 2 - InfluxDB	OK	05-20-2024 01:45:24	1d 3h 49m 41s	1/3	HTTP OK: HTTP/1.1 200 OK - 823 bytes in 0.007 second response time
	HW3 - Test 1 - Nodes	OK	05-20-2024 01:47:57	1d 7h 27m 8s	1/3	OK - Connection with nodes exists
	master-service	OK	05-20-2024 01:52:12	6d 15h 32m 53s	1/3	PING OK - Packet loss = 0%, RTA = 1.31 ms
DevOpscourse-internal-nagiosstudent-worker1	HW3 - Test 2 - NGINX	CRITICAL	05-20-2024 01:50:53	1d 2h 33m 22s	3/3	(No output on stdout) stderr: connect to address 172.17.89.216 port 5666: Connection refused
	worker-service	OK	05-20-2024 01:47:12	1d 8h 29m 57s	1/3	PING OK - Packet loss = 0%, RTA = 1.41 ms
DevOpscourse-internal-nagiosstudent-worker2	HW3 - Test 2 - NGINX	OK	05-20-2024 01:50:38	0d 23h 44m 27s	1/3	OK - NGINX on pods works
	worker-service	OK	05-20-2024 01:45:14	1d 9h 29m 51s	1/3	PING OK - Packet loss = 0%, RTA = 15.91 ms
DevOpscourse-internal-stnt-master	HW1 Flask app	PENDING	N/A	0d 0h 0m 2s+	1/3	Service check scheduled for Mon May 20 01:57:58 UTC 2024
	HW2 - Test 1 - Flask app in Docker	PENDING	N/A	0d 0h 0m 2s+	1/3	Service check scheduled for Mon May 20 02:00:54 UTC 2024
	HW2 - Test 2 - InfluxDB	PENDING	N/A	0d 0h 0m 2s+	1/3	Service check scheduled for Mon May 20 02:03:50 UTC 2024
	HW3 - Test 1 - Nodes	PENDING	N/A	0d 0h 0m 2s+	1/3	Service check scheduled for Mon May 20 01:58:18 UTC 2024
	master-service	PENDING	N/A	0d 0h 0m 2s+	1/3	Service check scheduled for Mon May 20 02:01:14 UTC 2024
localhost	Current Load	OK	05-20-2024 01:50:52	11d 9h 59m 16s	1/4	OK - load average: 0.00, 0.01, 0.00
	Current Users	OK	05-20-2024 01:54:58	20d 4h 33m 25s	1/4	USERS OK - 0 users currently logged in
	HTTP	OK	05-20-2024 01:54:59	20d 4h 32m 48s	1/4	HTTP OK: HTTP/1.1 200 OK - 10945 bytes in 0.001 second response time
	PING	OK	05-20-2024 01:54:51	20d 4h 37m 10s	1/4	PING OK - Packet loss = 0%, RTA = 0.07 ms
	Root Partition	OK	05-20-2024 01:54:59	20d 4h 36m 33s	1/4	DISK OK - free space: / 12949 MiB (65.91% inode=95%);
	SSH	OK	05-20-2024 01:54:59	20d 4h 35m 55s	1/4	SSH OK - OpenSSH_8.9p1 Ubuntu-3ubuntu0.6 (protocol 2.0)
	Swap Usage	CRITICAL	05-20-2024 01:54:59	20d 4h 32m 18s	4/4	SWAP CRITICAL - 0% free (0 MB out of 0 MB) - Swap is either disabled, not present, or of zero size.
	Total Processes	OK	05-20-2024 01:53:55	20d 4h 34m 40s	1/4	PROCS OK: 35 processes with STATE = RSZDT

Results 1 - 26 of 26 Matching Services

Figure 18. Example of Nagios Core UI after creating a student machine

HW1 Flask app	CRITICAL	(No output on stdout) stderr: connect to address 172.17.91.168 port 5666: Connection refused
HW2 - Test 1 - Flask app in Docker	CRITICAL	connect to address 172.17.91.168 and port 5001: Connection refused
HW2 - Test 2 - InfluxDB	CRITICAL	connect to address 172.17.91.168 and port 8086: Connection refused
HW3 - Test 1 - Nodes	CRITICAL	(No output on stdout) stderr: connect to address 172.17.91.168 port 5666: Connection refused
master-service	OK	PING OK - Packet loss = 0%, RTA = 89.05 ms

Figure 19. Example of student machine state on the Nagios Core UI before installing NRPE on it

HW1 Flask app	CRITICAL	CRITICAL - IP does not dynamically update or webapp does not run
HW2 - Test 1 - Flask app in Docker	CRITICAL	connect to address 172.17.91.168 and port 5001: Connection refused
HW2 - Test 2 - InfluxDB	CRITICAL	connect to address 172.17.91.168 and port 8086: Connection refused
HW3 - Test 1 - Nodes	CRITICAL	CRITICAL - Connection with nodes cannot be confirmed
master-service	OK	PING OK - Packet loss = 0%, RTA = 1.18 ms

Figure 20. Example of student machine state on the Nagios Core UI after installing NRPE on it

to check the student machine. After checking, the system will reach a state similar to Figure 19. These results indicate that the student machine does not respond to the Nagios Core checks. This is caused due to not having yet installed and configured the NRPE agent on the student machine.

To address this issue, the NRPE agent must be installed on the student machine using the instructions provided in Section 4.3. After installing and waiting for a bit, the results will still be CRITICAL and red, but this time, the messages in the last column are different (refer to Figure 20). Two of them are still unchanged because for them, Nagios Core returns a default message. On the other hand, the other two CRITICAL status checks return a message. These messages are caught in the student machine after executing the check command defined. The check command is configured to give a message when the service is not working properly. This indicates that the system can contact the student machine and execute a script on it properly.

The next task is to do the homework. Something should be noted when doing this. When the student creates the app, they create a Python virtual environment where they store the installed packages. These packages are not inherently accessible to the Nagios user (which is the user that performs the checks). To give access to the packages, the virtual environment configuration file *pyvenv.cfg* should be updated by changing the value *include-system-site-packages* from *false* to *true*, such as shown in Figure 21.

Using this approach, the NRPE agent can execute the Flask application and report back its status, which in this case, is OK (refer to Figure 22).

This demonstrates a good example of executing complicated scripts on the student's virtual machine and getting the results of them automatically.

Figure 21. Changing the pyvenv.cfg file with before and after the change (false is before, true is after)

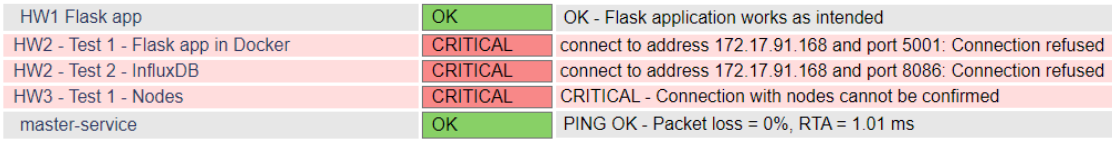


Figure 22. Example of student machine state on the Nagios Core UI after finishing the first homework

5.2 Homework 2: Working with Docker

The second homework is about introducing Docker and containerisation to the students. The students will first deploy their Flask application using Docker, then they will delete it, deploy an InfluxDB database and finally deploy the Flask application again, but this time, using the InfluxDB database they created.

This homework is chosen to demonstrate a less intrusive way of checking the tasks.

The idea is to check if the containers deployed by the students are running by checking the services they are supposed to host using the simple *check_http* command from the Nagios Core. This plugin will try to ping the HTTP website specified and return if it is successful.

Before deploying the first application, the state of the machine is as in Figure 22. After deploying the application (note the port has been changed to 5001 instead of 5000 to avoid collision between the first and second homework), the UI should show the first task of homework 2 as OK (refer to Figure 23).

After deleting this container, the state should revert to what was displayed in Figure 22. As expected, this is what happens (refer to Figure 24).

The next step is to deploy the InfluxDB database container. Since InfluxDB also

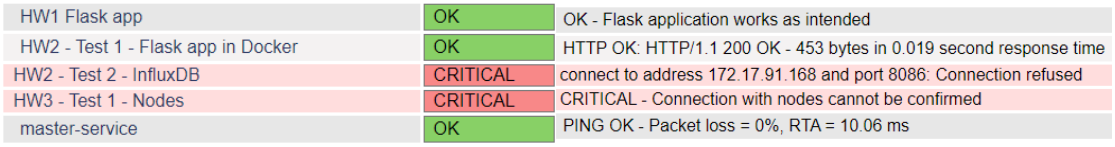


Figure 23. Example of student machine state on the Nagios Core UI after finishing the first task in the second homework

HW1 Flask app	OK	OK - Flask application works as intended
HW2 - Test 1 - Flask app in Docker	CRITICAL	connect to address 172.17.91.168 and port 5001: Connection refused
HW2 - Test 2 - InfluxDB	CRITICAL	connect to address 172.17.91.168 and port 8086: Connection refused
HW3 - Test 1 - Nodes	CRITICAL	CRITICAL - Connection with nodes cannot be confirmed
master-service	OK	PING OK - Packet loss = 0%, RTA = 10.06 ms

Figure 24. Example of student machine state on the Nagios Core UI after removing the container made in the first task of the second homework

HW1 Flask app	OK	OK - Flask application works as intended
HW2 - Test 1 - Flask app in Docker	CRITICAL	connect to address 172.17.91.168 and port 5001: Connection refused
HW2 - Test 2 - InfluxDB	OK	HTTP OK: HTTP/1.1 200 OK - 823 bytes in 0.005 second response time
HW3 - Test 1 - Nodes	CRITICAL	CRITICAL - Connection with nodes cannot be confirmed
master-service	OK	PING OK - Packet loss = 0%, RTA = 10.06 ms

Figure 25. Example of student machine state on the Nagios Core UI after deploying the InfluxDB container

hosts a UI as a web service, then it is possible to ping it like with task 1. After deploying, the state of the student machine should show that in homework 2, only the second task is OK. When checking on the Nagios Core UI, this is the case (refer to Figure 25).

As the last task, it is required to deploy the Flask application, but this time using the data from the InfluxDB database. Therefore, Nagios Core UI should show both tasks of the second homework in the state OK. This is the case in reality as well, since the result of this deployment is as in Figure 26.

These 2 tests are enough because it is easier for the student to follow the guide rather than to fake these web services. The fakes would need to be constantly running, similarly to the containers. Disconnecting from the SSH session will close most processes created on the SSH session. Therefore, creating fake web services would require the student to use either *Systemctl*, Docker or something similar. Docker is most likely the easiest approach and to fake it, they will need to study it, which means the goal of the practice session would be fulfilled. Additionally, the homework instructions are very clear, meaning that most likely, the process of faking is more complex than actually following the guide. Therefore, this test is a good example of how to approach grading in a simpler and less intrusive way, while checking that the student learned new things

HW1 Flask app	OK	OK - Flask application works as intended
HW2 - Test 1 - Flask app in Docker	OK	HTTP OK: HTTP/1.1 200 OK - 453 bytes in 0.009 second response time
HW2 - Test 2 - InfluxDB	OK	HTTP OK: HTTP/1.1 200 OK - 823 bytes in 0.019 second response time
HW3 - Test 1 - Nodes	CRITICAL	CRITICAL - Connection with nodes cannot be confirmed
master-service	OK	PING OK - Packet loss = 0%, RTA = 1.17 ms

Figure 26. Example of student machine state on the Nagios Core UI after deploying both containers

from the practice session.

6 Limitations

There are some additional limitations to this system, which have not been mentioned before. They are the following:

- The system is designed for running the built-in Nagios Core and NRPE plugins along with the custom *run_script* plugin as instructed and showing the results on the UI. Additional use cases are not guaranteed.
- It is impossible to guarantee a 100% working system, and therefore, there might be cases, when something breaks, which might be caused due to external causes or bugs in Nagios and NRPE themselves.
- The system's automatic host updating script works in the current version (as of May 2024) of ETAIS and uses an API key to get info about the other projects (the students' projects). The API key should be from a user with sufficient access to see all the projects (which are needed to be graded) and their contents, otherwise it cannot be guaranteed to update the configurations correctly.
- The system has been tested on Linux Ubuntu Version 22.04, it is not guaranteed to work on other versions.
- The system has been tested to run on the *m2.tiny* ETAIS flavor, which gives 2 CPU cores and 4GB of RAM.
- The grading and target systems should be able to contact each other via the hostaddresses (IP addresses) defined in the configurations. If the grading system is built in a LAN at a private network and the target system is on a VPC, then communication between the 2 machines cannot be guaranteed.

7 Conclusion

I, Mihkel Hani, created this project. Nagios Core and NRPE are tools that existed beforehand and I used them to aid with the creation of such a system. The supervisor, Chinmaya Kumar Dehury, provided me with the script *check_VMs_status.py* which I modified for the current use case.

The result of this work is an automated grading system, which is easy to configure and supports easily creating new service checks. It is going to be used for the University of Tartu's course "DevOps: Automating Software Delivery and Operations" (LTAT.06.016) to streamline the students', teaching assistants' and teachers' experience. The system is able to continuously grade students' works, which not only helps avoid long waiting times for feedback but also gives time for the teachers and teaching assistants to focus on answering questions and helping students. The system automatically detects when a new machine has been made in the ETAIS platform and adds it to the list. The installation for the students has been made especially easy, so that they do not need to waste their focus on installing the system and can use their energy on solving the tasks given to them.

The system can be further improved with many additions. It was built with documentation and future updating in mind. Future possibilities include:

- Automatically configuring the students' machines.
- Creating a new student UI, where the student can log in using their University credentials, so they can have a cleaner overview of how they are doing.
- Incorporate a system that checks if something has been done too fast and if yes, compares it to others (like somebody created a script too fast and if it matches somebody else's script, it might be plagiarised).

In conclusion, this project was a great success.

References

[1] "Information technology — Vocabulary — Part 36: Learning, education and training." https://www.iso.org/obp/ui#iso:std:iso-iec:2382:-36:ed-3:v1:en. Accessed: 2024-05-05.

[2] "DevOps: Automating Software Delivery and Operations." https://courses.cs.ut.ee/2023/DevOps/fall. Accessed: 2024-05-05.

[3] "Information technology — Cloud computing — Common technologies and techniques." https://www.iso.org/obp/ui#iso:std:iso-iec:ts:23167:ed-1:v1:en:term:3.10. Accessed: 2024-05-05.

[4] "Resources." https://etais.ee/resources_ut/. Accessed: 2024-05-05.

[5] "About." https://etais.ee/about/. Accessed: 2024-05-05.

[6] "Prices." https://etais.ee/prices/. Accessed: 2024-05-06.

[7] "What is latency? | How to fix latency." https://www.cloudflare.com/learning/performance/glossary/what-is-latency/. Accessed: 2024-05-06.

[8] "Top 10 Best DevOps Automation Tools We Recommend." https://itoutposts.com/blog/top-7-devops-automation-tools/. Accessed: 2024-05-05.

[9] "About." https://www.nagios.org/about/. Accessed: 2024-05-05.

[10] "Nagios Open Source." https://www.nagios.org/. Accessed: 2024-05-05.

[11] "Architecture." https://support.nagios.com/kb/article/nrpe-architecture-141.html. Accessed: 2024-05-05.

[12] "About github." https://docs.github.com/en/get-started/start-your-journey/about-github-and-git. Accessed: 2024-05-14.

[13] "CronJob." https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/. Accessed: 2024-05-06.

[14] L. Baumstark and E. Rudolph, "Automated online grading for virtual machine-based systems administration courses," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, (New York, NY, USA), p. 477–482, Association for Computing Machinery, 2013.

[15] "Lab 2 - ETAIS." https://sysadmin.cs.ut.ee/lab2/. Accessed: 2024-05-05.

[16] "Error: Permission denied (publickey)." https://docs.github.com/en/authentication/troubleshooting-ssh/error-permission-denied-publickey. Accessed: 2024-05-06.

[17] "Nagios core installation guide 1." https://www.nagios.org/downloads/nagios-core/. Accessed: 2024-05-14.

[18] "Nagios core installation guide 2." https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/quickstart.html. Accessed: 2024-05-14.

[19] "Nagios core installation guide 3." https://support.nagios.com/kb/article/nagios-core-installing-nagios-core-from-source-96.html. Accessed: 2024-05-14.

[20] "Nrpe agent installation guide 1." https://assets.nagios.com/downloads/nagioscore/docs/nrpe/NRPE.pdf. Accessed: 2024-05-14.

[21] "Nrpe agent installation guide 2." https://support.nagios.com/kb/article/nrpe-how-to-install-nrpe-8.html. Accessed: 2024-05-14.

[22] "Nrpe agent installation guide 3." https://support.nagios.com/kb/article.php?id=515. Accessed: 2024-05-14.

Appendix

I. Source Code

This project's public source code, which consists of the documentation and the target systems' installation script, can be found in the LabEvalAutomation-public GitHub repository (https://github.com/DevOps-TextBook/LabEvalAutomation-public/). The private content of the code, such as the installation scripts for the grading system, can be made available on a demand basis.

II. Acknowledgements

I would like to express my deepest appreciation to my supervisor, Chinmaya Kumar Dehury, for Their invaluable supervision, support and tutelage to me. Additionally, I could not have created this thesis without the teachings that the University of Tartu has given me during my Bachelor's studies and would like to thank Them for all of the hard work.

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Mihkel Hani**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Automated Grading System: The DevOps course Use Case,

   supervised by Chinmaya Kumar Dehury.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mihkel Hani
15/05/2024