DATA2410-1 22V Datanettverk og Skytjenester

GROUP Portfolio Assignment 2 - Docker and Zabbix Real Use-Case

Ulrik Bakken Singsaas | s351917
Adrian Bakstad | s341507
Thea Emilie Haugen | s351879
Andrea Bjørge | s344175

April 25, 2022

Contents

1. Introduction

1.1. Our project directory

At the beginning of the project we set up a group project directory called `portfolio2`. In this directory we decided to sort our files into different sub-folders for structure, backup and easy access purposes. The files were sorted based on their functionality and contents. Below is a list of the different sub-folders that were of significance to our project. Many of these sub-folders will be referenced in the report.

docker: files pertaining to docker functionality

docs: files pertaining to documentation and explanation of our project

configs: files that aren't in use, but are kept as backup - configs/intel1: redundant docker files for VMs on intel1 - configs/mysql: old mysql configuration files - configs/nginx: old nginx configuration files - configs/zabbix: old Zabbix configuration files

1.2. Virtual Machines with VirtualBox

Originally, we attempted to use docker containers on the intel1-server to implement our solution. However, the server ran out of storage space, so we created virtual machines through VirtualBox as a substitute.

The first thing we needed was a VM running Ubuntu Focal Fossa. Cloning this VM would let us create the 3 virtual machines needed for the assignment, each with 4GB of RAM and 10GB of disk space. We started by downloading the image for Ubuntu Focal Fossa (20.04) from: https://releases.ubuntu.com/20.04/ubuntu-20.04.4-desktop-amd64.iso and creating VM1.

We started by setting up VM1 on a bridged network. The reason we started with VM1 was to make sure that it was working. We created VM2 by cloning VM1 and changing the mac address. We cloned VM1 one more time to create VM3. By changing the mac addresses we ensured that each VM had their own local IP on the bridged network. By doing it this way, we made sure that all three VMs could communicate with each other, whilst also being able to communicate with the host machine.

The architecture diagram in the assignment description can be interpreted to mean that we should use an internal network for all of the VMs whilst giving VM2 a second bridged network adapter. This would ensure that only the nginx proxy could reach the outside of the internal VM network. Since the assignment didn't specify what network method to use for th VMs, we decided against this, because this would make our assignment more complicated than necessary.
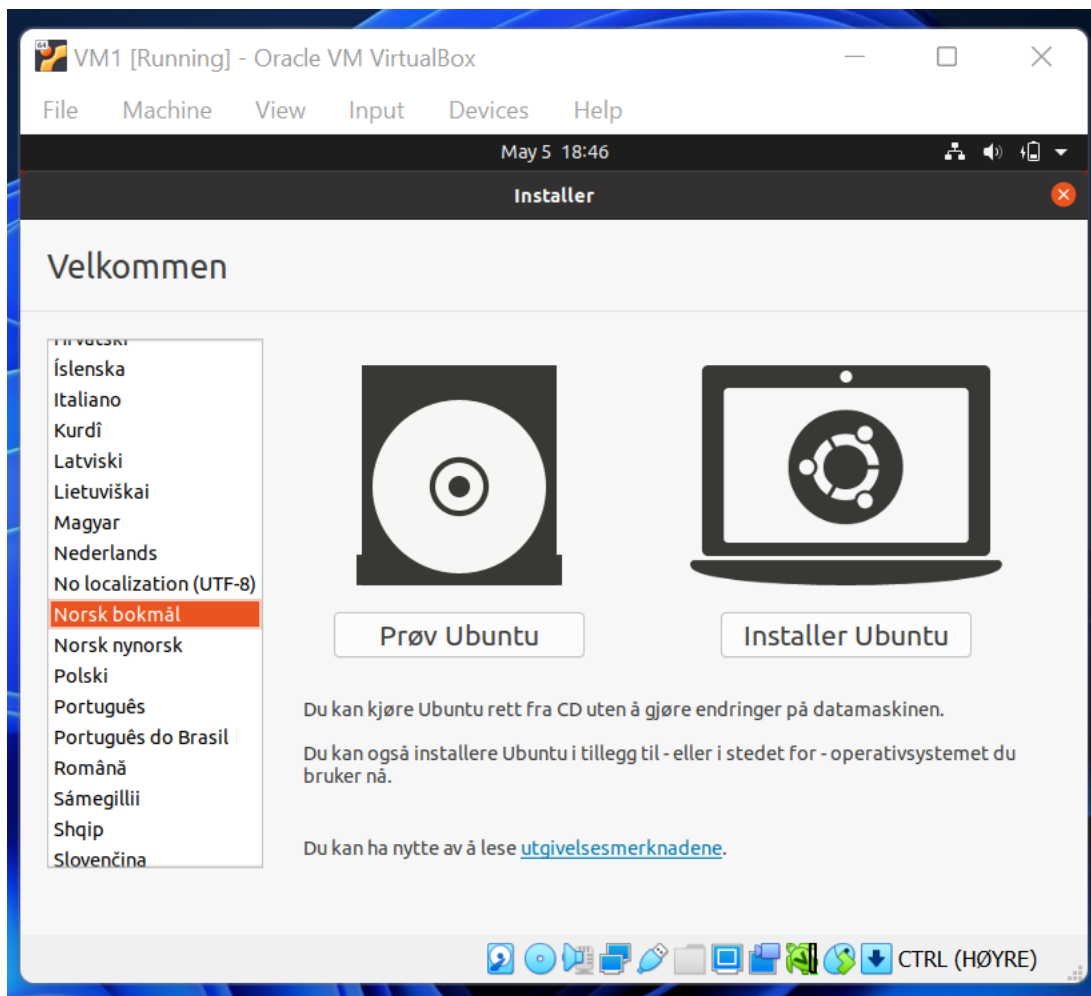
Figure 1: Ubuntu installation screen

2. VM1: Docker containers setup - Docker Compose Setup

After setting up the three VMs, we used the file docker-compose.yml, from the docker folder in our project directory, to set up the four docker containers with the required config instructions for the assignment within VM1. This file can be found in the docker folder in our project directory. The first step in setting up the docker containers was to install docker on VM1.

We used the following command to install docker.

```
sudo apt-get install -y docker-compose
```

We used the auto generated docker files to set up the docker containers, but we made some minor adjustments by setting the environment variables for these files before they were generated. The environment variables were created outside the docker-compose.yml file, but later references in the file. We created volume links as external volumes to make it possible to edit the internal volume files outside the docker containers via the docker volume functionality. This was possible because the volume files outside the docker containers are synchronized with the volume files we mapped them to inside the docker containers. The docs volume was used to get the .sql file to create the server. Setting up volumes for outside access made debugging easier while working on the project. For example we utilized this setup to verify if the environment variables in the docker-compose.yml file was correctly written in the zabbix_server.conf config file, while remaining outside the docker container containing the Zabbix Server.

The following block of code describes how we set up the volumes according to the description above.

```
sudo docker volume create mysql-server-data
sudo docker volume create zabbix-server-config
sudo docker volume create zabbix-web-config
sudo docker volume create zabbix-agent-config
sudo docker volume create docs
```

In the block below, we display our docker compose file docker-compose.yml from the docker folder in our project directory.

```yaml
# Docker Container setup for VM 1
version: "3.0"
services:
  # mysql container setup
  mysql-server:
    image: haakco/mysql80
    ports:
      - 3306
    hostname: mysql-server
    restart: unless-stopped
    volumes:
      - mysql-server-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=123
      - MYSQL_DATABASE=zabbix
      - MYSQL_USER=zabbix
      - MYSQL_PASSWORD=123
    cap_add:
      - SYS_NICE
    networks:
      zabbix-net:
        ipv4_address: 172.200.1.1
        # not really supposed to have two commands put putting it on one line broke things and this seems to wo
        # tried this, but it did not work:
        # command: bash -c "--default-authentication-plugin=mysql_native_password && --datadir=/var/lib/mysql/d
    command: --default-authentication-plugin=mysql_native_password
    command: --datadir=/var/lib/mysql/data
```

```yaml
# zabbix server setup
zabbix-server:
  image: zabbix/zabbix-server-mysql
  ports:
    - 10051:10051
  hostname: zabbix-server
  restart: unless-stopped
  volumes:
    - zabbix-server-config:/etc/zabbix
    - docs:/usr/share/doc/
  environment:
    - DB_SERVER_HOST=mysql-server
    - MYSQL_DATABASE=zabbix
    - MYSQL_USER=zabbix
    - MYSQL_PASSWORD=123
  depends_on:
    - mysql-server
  networks:
    zabbix-net:
      ipv4_address: 172.200.1.2

# zabbix web container
zabbix-web:
  image: zabbix/zabbix-web-nginx-mysql
  ports:
    - 80:8080
  hostname: zabbix-web
  restart: unless-stopped
  volumes:
    - zabbix-web-config:/etc/zabbix
  environment:
    - DB_SERVER_HOST=mysql-server
    - MYSQL_DATABASE=zabbix
    - MYSQL_USER=zabbix
    - MYSQL_PASSWORD=123
    - ZBX_SERVER_HOST=zabbix-server
  depends_on:
    - mysql-server
    - zabbix-server
  networks:
    zabbix-net:
      ipv4_address: 172.200.1.3

# zabbix agent container
zabbix-agent:
  image: zabbix/zabbix-agent
  ports:
    - 10050:10050
  hostname: zabbix-agent
  restart: unless-stopped
  volumes:
    - zabbix-agent-config:/etc/zabbix
  environment:
    - ZBX_SERVER_HOST=zabbix-server
  depends_on:
```

```
      - zabbix-server
    networks:
      zabbix-net:
        ipv4_address: 172.200.1.4

# custom network for the containers
networks:
  zabbix-net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.200.1.0/16

# volumes for the containers
volumes:
  mysql-server-data:
    external: true
  zabbix-server-config:
    external: true
  zabbix-web-config:
    external: true
  zabbix-agent-config:
    external: true
  docs:
    external: true
```

We used the following command on VM1 to start the docker containers with the `docker-compose.yml`.

```
sudo docker-compose up
```

After the docker containers were up and running, we decided to set up host profiles for active and passive checks between the zabbix agent and server in the docker stack. This was to ensure that everything connected properly. The web frontend is hosted on VM1 port 80 as per the assignment description, so to reach it, we simply typed the address in a web browser. In later steps of the assignment, we set up an nginx proxy that allowed us to reach the frontend trough the localhost-address.

| zabbix-agent | Items 66 | Triggers 34 | Graphs 13 | Discovery 3 | Web | |
| zabbix_server | Items 67 | Triggers 34 | Graphs 13 | Discovery 3 | Web | 172.200.1.4:10050 |

| Linux by Zabbix agent active | Enabled | | None |
| Linux by Zabbix agent | Enabled | ZBX | None |

Figure 2: Image showing that the zabbix-agent and zabbix-server is working

Here is a screenshot of our docker compose log, showing that all the checks except for one is working between the agent and server. Our thoughts was that this one check from the template probably wasn't suited for being run in a docker environment as some things can be different there. We decided to cut this image in two and put them ih two rows to make it easier to read in a standing page .pdf format

So this should fully explain how we did the first part of this assignment, how we installed docker, how we configured our docker-compose stack and made our docker bridge network inside of VM1, while also going a little further by using the frontend to set up the hosts there.'

Figure 3: Logs from docker compose after setting up hosts on frontend

3. VM2 and VM3: Install zabbix-agent and zabbix-proxy

3.1. VM2

We followed this guide to complete task 1 in part III of the assignment description: https://bestmonitoringtools.com/install-zabbix-proxy-on-ubuntu/ There were a few things we did differently from the guide. These will be described below.

3.1.1. Installing Zabbix Proxy

We started by installing `Zabbix Proxy` on VM2 with the following commands:

NB: These links are not the same given in the tasks as using the release packages gave a lot of problems with wrong versions, sometimes we would get version 4 or 5, and sometimes a 6.2 beta version, whilst all we needed was the 6.0.x versions as these were the ones that the docker compose stack received. We got our downloads from the zabbix repo listed in our references

```
apt-get install wget

wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/
  zabbix-release_6.0-1%2Bubuntu20.04_all.deb

dpkg -i zabbix-release_6.0-1+ubuntu20.04_all.deb

# needed this as well since we got the wrong version (6.2 beta) of the proxy by just having the release package
wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix/
  zabbix-proxy-mysql_6.0.1-1%2Bubuntu20.04_amd64.deb

dpkg -i zabbix-proxy-mysql_6.0.1-1+ubuntu20.04_amd64.deb

apt-get install -f

apt-get install zabbix-proxy-mysql

apt-get install zabbix-sql-scripts
```

The only notable difference between the guide and what we did in this part of the task, is that we used Zabbix-proxy version 6.0.1 instead of Zabbix-proxy version 6.1, because the assignment descriptions asks us to use version 6.1.1

3.1.2. Configuring MariaDB database for the proxy to use

After we finished installing `Zabbix Proxy`, we installed and configured the database, using `MariaDB`, according to both the guide mentioned above, and the assignment description.

The following block of code describes the installation of `MariaDB` on VM2:

```
sudo apt install software-properties-common -y

curl -LsS -O https://downloads.mariadb.com/MariaDB/mariadb_repo_setup

sudo bash mariadb_repo_setup --mariadb-server-version=10.6

sudo apt update

sudo apt -y install mariadb-common mariadb-server-10.6 mariadb-client-10.6
```

After installing `MariaDB` we began configuring the database by running the following commands to start and enable MariaDB, and configure it to start on boot:

```
sudo systemctl start mariadb

sudo systemctl enable mariadb
```

The next step in the configuration of the database was to reset the password. We did that with the commands in the following code block: The following code blocks set the password for new **root password** = 123.

```
sudo mysql_secure_installation

Enter current password for root (enter for none): Press Enter

Switch to unix_socket authentication [Y/n] y

Change the root password? [Y/n] y

New password: <Enter root DB password>

Re-enter new password: <Repeat root DB password>

Remove anonymous users? [Y/n] y

Disallow root login remotely? [Y/n] y

Remove test database and access to it? [Y/n] y

Reload privilege tables now? [Y/n] y
```

The only notable difference between the guide and what we did was that we set the root password to '123', in stead of 'rootDBpass.

After we set the root password, it was time to create the database by running the commands in the following block of code:

```
sudo mysql -uroot -p'123' -e "create database zabbix_proxy character set utf8mb4 collate utf8mb4_bin;"

sudo mysql -uroot -p'123' -e "grant all privileges on *.* to zabbix@localhost identified by 'zabbixDBpass';"
```

The last step in the configuration of the database was to import the initial schema and data with the following command:

```
sudo cat /usr/share/doc/zabbix-sql-scripts/mysql/proxy.sql | mysql -uzabbix -p'zabbixDBpass' zabbix_proxy
```

In the installation and configuration of the database, we followed the guide quite exactly. Therefore, there are very few differences between what we did to install and configure the database, and what is stated in the installation guide.

3.1.3. Configuring Zabbix Proxy

Once the installation and configuration of the database was complete, it was time to configure the `Zabbix Proxy`. The first step when configuring the `Zabbix Proxy` was to open the config file with the following command:

```
sudo gedit /etc/zabbix/zabbix_proxy.conf
```

In the file we changed the following values:

```
DBPassword=zabbixDBpass
ConfigFrequency=100
Server=192.168.50.95
Hostname=Zabbix Proxy
DBName=zabbix_proxy
DBUser=zabbix
```

After editing the necessary values, we saved and exited the file. We set the `ConfigFrequency` to be 100 seconds. This parameter determines how often the proxy retrieves data from the configuration file, and is useful to cut down on the waiting time between updates on the status of the `Zabbix Proxy`. The notable differences between our config file, and the config file in the guide is that we have a different IP address for the server, and a different hostname for the proxy itself.

3.1.4. Starting and enabling the Zabbix Proxy

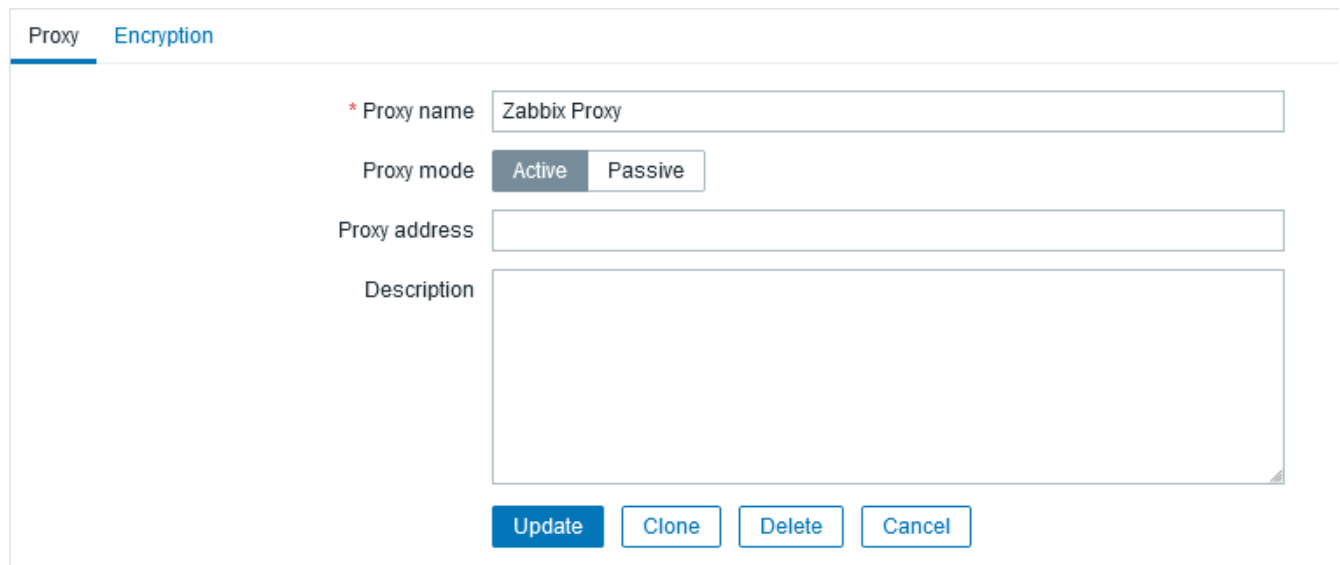Next, we started the `Zabbix Proxy` and enabled it to boot on startup with the following commands:

```
# makes the proxy start by default
sudo systemctl enable zabbix-proxy

# not really needed but I like to do this just in case
sudo systemctl start zabbix-proxy
```

It is important to take note of the Hostname in the zabbix-proxy config file. We need it in order to connect the proxy to the server using the web frontend. While in the zabbix-proxy config file, it is also important to make sure that the DBPassword is set to the correct value.

3.1.5. Registering Zabbix Proxy in the Zabbix frontend

Quickly documenting how we did the frontend setup of the proxy after, doing the config on VM2, and also an image verifying that it was connected

Name ▲	Mode	Encryption	Compression	Last seen (age)	Host count	Item count
Zabbix Proxy	Active	None PSK	On	3s	1	73

Figure 4: Image showing that the zabbix-proxy is connected

3.2. VM 3. Zabbix Agent installation and setup

The following code block must be run as root on VM3 to install the `Zabbix-agent`.

```
wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/
  zabbix-release_6.0-1%2Bubuntu20.04_all.deb

sudo dpkg -i zabbix-release_6.0-1+ubuntu20.04_all.deb

#had to use this one for right version
wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix/
  zabbix-agent_6.0.1-1%2Bubuntu20.04_amd64.deb

sudo dpkg -i zabbix-agent_6.0-1+ubuntu20.04_amd64.deb

sudo apt-get install -f

sudo apt-get install zabbix-agent
```

The following code block creates the psk encryption key.

```
openssl rand -hex 32 > zabbix_agent.psk

cat zabbix_agent.psk
e8126679667a8594bc8d3d76121b6ba2a5fb4b6d41bea2cd62190c163fbc6c6b
```

The following code block moves the psk encryption key to /opt/zabbix folder.

```
sudo mkdir /opt/zabbix

sudo chmod 777 /opt/zabbix

sudo mv zabbix_agent.psk /opt/zabbix/
```

Used these commands to edit the `zabbix-agent.conf` file

```
sudo vim /etc/zabbix/zabbix_agentd.conf
```

The following code block contains the lines we changed in the `zabbix-agent.conf` file; to enable psk encryption.

```
TLSConnect=psk
TLSAccept=psk
TLSPSKIdentity=cbt_psk_01,
TLSPSKFile=/opt/zabbix_agent.psk
# the local ip of our bridged networked VM2
Server=192.168.50.247
```

Lastly we start the agent with the following command:

```
sudo systemctl restart zabbix-agent
```

See point/heading 5 to see how we configured this in the front end.

4. VM2: Nginx proxy

We started by installing nginx with the following commands.

```
sudo apt-get update

sudo apt-get install nginx
```

Once nginx was installed, we disabled the default virtual host by unlinking it, using the following command.

```
sudo unlink /etc/nginx/sites-enabled/default
```

To add the new configurations to the proxy, we moved to the `sites-available` directory to create our new file.

```
cd /etc/nginx/sites-available/
nano reverse-proxy.conf
```

Here we created a new configuration file for the nginx-proxy. These configurations makes sure that the nginx-proxy listens on port 8080, and redirects all incoming traffic from that port to the `Zabbix-server` using `proxy_pass` followed by the Zabbix-server's IP address and port number.

```
server {
    listen  8080;
    server_name localhost;

    location / {
        proxy_pass http://192.168.50.95:80;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

To complete the proxy, we activate the directives by linking to `/sites-enabled/` using the following command.

```
sudo ln -s /etc/nginx/sites-available/reverse-proxy.conf /etc/nginx/sites-enabled/reverse-proxy.conf
```

Lastly, to see if it worked, we ran an nginx configuration test and restarted the service.

```
sudo systemctl configtest nginx
 * Testing nginx configuration                                        [ OK ]

sudo systemctl restart nginx
 * Restarting nginx nginx                                             [ OK ]
```

This verifies that nginx works as intended.

Image of nginx-proxy being accessed from host machine via VM2 local IP in bridged network on port 8080. This isn't strictly localhost:8080 based on the diagram.

Hostnames on all VMs where `ubuntu1` because vm2 and vm3 where created by cloning vm1.
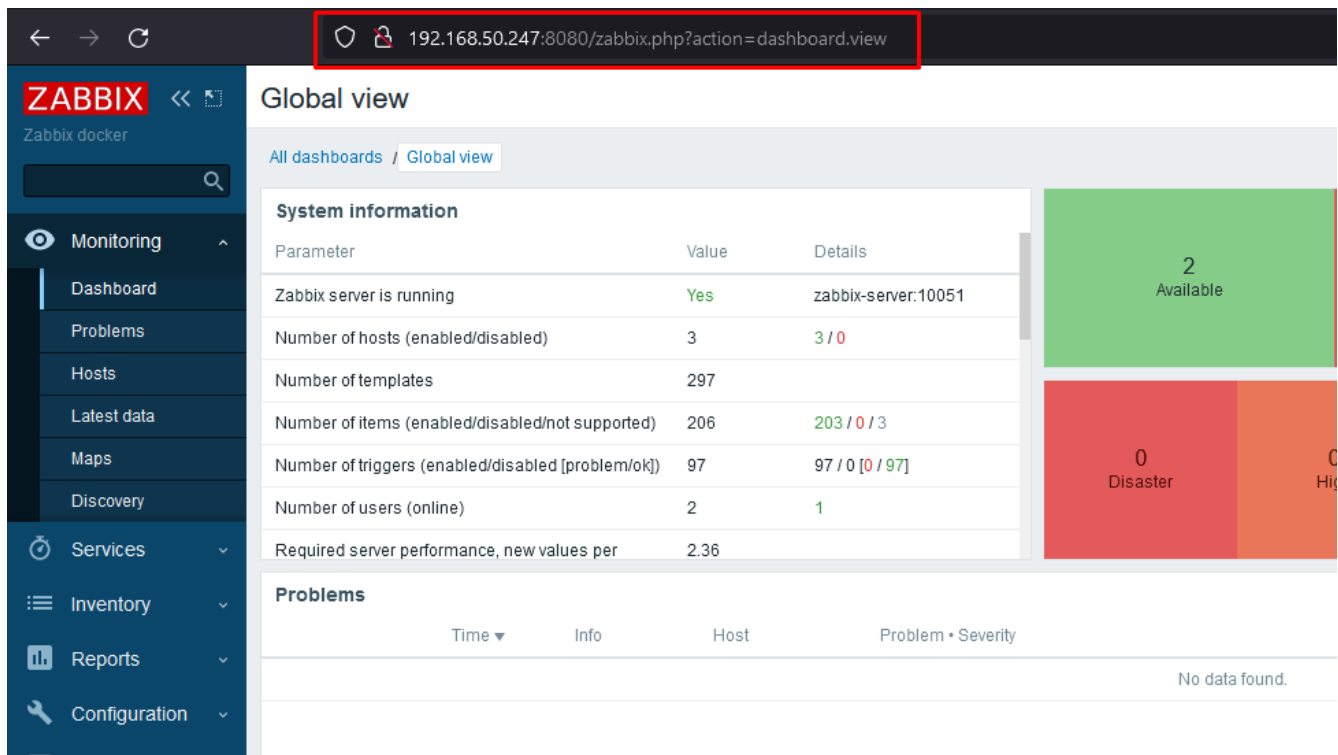
Figure 5: Image showing zabbix frontend from nginx proxy

Figure 6: Image showing local ip of vm2

5. VM1: Zabbix frontend

To access the Zabbix frontend, we connected to the nginx-proxy on VM2 via its local IP and port 8080 as specified. This redirected us to the VM1 zabbix-web docker container. Once logged in to the Zabbix frontend, we added the host according to the descriptions, made the items as per point a) and b) and lastly the triggers as per point c) and d)

Name ▲	Items	Triggers	Graphs	Discovery	Web	Interface
zabbix-agent	Items 66	Triggers 34	Graphs 13	Discovery 3	Web	
zabbix_server	Items 67	Triggers 34	Graphs 13	Discovery 3	Web	172.200.1.4:10050
zabbix_server_agent_vm3	Items 73	Triggers 29	Graphs 14	Discovery 3	Web	192.168.50.151:10050

Proxy	Templates	Status	Availability	Agent encryption
	Linux by Zabbix agent active	Enabled		None
	Linux by Zabbix agent	Enabled	ZBX	None
Zabbix Proxy	Linux by Zabbix agent	Enabled	ZBX	PSK None PSK CERT

Figure 7: Image of our host setup with VM3 agent, split in two for easier viewing on paper

We created a new template named zabbix-monitoring in the zabbix-monitoring host group:

5.1. Items

We created the item for total disk space usage in the directory `/var` with interval of 1 hour:

```
fsv.fs.size[/var,used]
```

We created an that will monitor the docker process usage with interval of 1 minute:

```
proc.cpu.util[dockerd]
```

		Name ▲	Triggers	Key	Interval	History
☐	•••	Docker process usage		proc.cpu.util[dockerd]	1m	90d
☐	•••	Total Used Disk Space		vfs.fs.size[/var,used]	60m	90d

Figure 8: Image showing that the items are created

5.2. Triggers

We created a trigger that triggers when uptime is longer than 240 days:

```
last(/zabbix_server_agent_vm3/system.uptime)>240d
```

All hosts / zabbix_server_agent_vm3   Enabled   ZBX   Items 73   Triggers 28   Graphs 14   Discovery rules 3   Web scenarios

Trigger   Tags   Dependencies

* Name	uptime over 240 days
Event name	uptime over 240 days
Operational data	
Severity	Not classified Information Warning Average High Disaster
* Expression	last(/zabbix_server_agent_vm3/system.uptime)>240d Add
	Expression constructor
OK event generation	Expression Recovery expression None
PROBLEM event generation mode	Single Multiple
OK event closes	All problems All problems if tag values match

Figure 9: Image showing trigger uptime creation dialog

We created a trigger that triggers when disk I/O is higher than 20% average for 5 minutes:

```
avg(/zabbix_server_agent_vm3/system.cpu.util[,iowait],5m)>20
```

After making those triggers we made sure that the triggers where correctly created:

Figure 10: Image showing trigger disk io creation dialog

Figure 11: Image showing the uptime trigger is created

Figure 12: Image showing the disk io trigger is created

6. References

Canonical. (n.d.). *Ubuntu 20.04.4 LTS (Focal Fossa).* Ubuntu 20.04.4 lts (focal fossa). Retrieved May 12, 2022, from https://www.releases.ubuntu.com/20.04/

Downloads.mariadb.com. downloads.mariadb.com. (n.d.). Retrieved May 12, 2022, from https://downloads.mariadb.com/MariaDB/mariadb_repo_setup

Index of /zabbix/6.0/ubuntu/pool/main/z/zabbix/. Zabbix Official Repository. (n.d.). Retrieved May 12, 2022, from https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/

Lontons A. (2021, December 9). *Handy Tips #15: Deploying zabbix passive and active agents.* Zabbix Blog. Retrieved May 12, 2022, from https://blog.zabbix.com/handy-tips-15-deploying-zabbix-passive-and-active-agents/17696/

Zabbix proxy: Install on ubuntu 20.04 in 10 minutes! Best Monitoring Tools. (n.d.). Retrieved May 12, 2022, from https://bestmonitoringtools.com/install-zabbix-proxy-on-ubuntu/