# Practice M7: Management and Maintenance (CentOS)

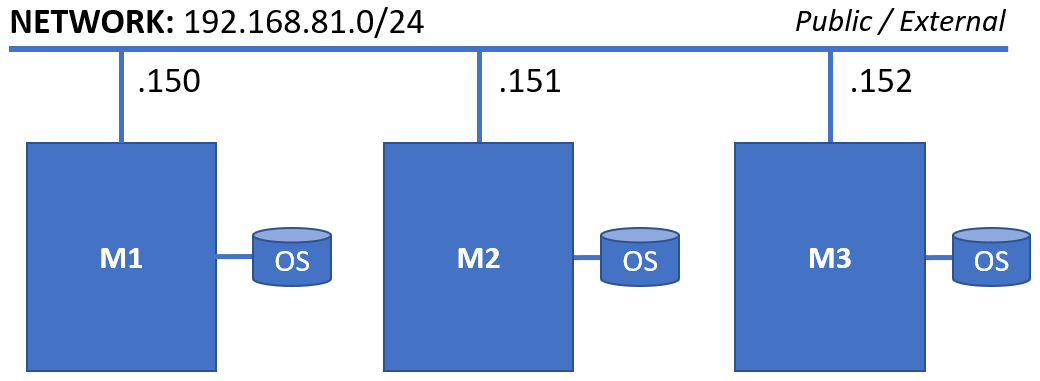
This practice assumes that you are working in an on-premise environment

All tasks can be achieved under different configurations (host OS and/or virtualization solution) with the appropriate adjustments

This practice is oriented towards **CentOS 8.4** and **Fedora Server 34**

## Part 1: Monitoring

For this part we will need an infrastructure like this one:



Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

### Conventional Tools

Log on to the **M1** machine with a regular user account

So far, we have used mainly tools, that give us information about the current state of the system

For example, should we want the current utilization of the memory in megabytes, we use

**free -m**

In a similar way, if we wanted to check how the disk space is utilize, we execute

**df -h**

For the average CPU utilization, we use

**uptime**

Of course, there are tools that combine two or more of these plus additional parameters. Some of them are available with the default installation, others we must install separately

For example, we have **top**, **htop**, **nmon**, **atop**, **btop**, **vmstat**, etc. (you may need to install EPEL and other repos)

All those utilities give us information about a chosen set of parameters with little or no history

As we cannot always monitor the system (or set of systems) by sitting in front of it and watching how the values of the parameters are changing, we must find another way

Usually, there are tools or even whole solutions, that offer us solution to our need. They give us centralization, aggregation, visualization, history, etc.

Some of them are oriented towards a single system, while others are, by design, created to cover multiple systems

### Sysstat

**Sysstat** is a tool (or set of tools) that collects and stores the performance data for a system

In order to install it, we must execute

**sudo dnf install sysstat**

It brings a set of tools including **iostat, pidstat, sar, sadf,** etc.

We can open the main configuration file and explore its content

**sudo vi /etc/sysconfig/sysstat**

Default configuration is more than okay, so let us close the file without modifying it

First, let us check what units have been installed

**systemctl list-units --all | grep sysstat**

We can see a few - three services and two timers

Let us enable and start one of the services

**sudo systemctl enable --now sysstat**

If we do the check again, we will see that the timer units got enabled as well

What do we have there, in the timer units? First, let us find where those are located

**sudo find / -type f -name "sysstat\*.timer"**

Now, let us check the **sysstat-collect.timer** unit

**cat /usr/lib/systemd/system/sysstat-collect.timer**

We can see that **every 10 minutes** the service with the same name **sysstat-collect.service** will be executed

For our purposes, we can change it to **1 minute** and reload / restart the service

**sudo systemctl daemon-reload**

**sudo systemctl restart sysstat**

Even though the **Unit=** directive is missing, by default the timer will launch a service with the same name

And the service on the other hand, contains instructions to store the picture of the state of the system

**cat /usr/lib/systemd/system/sysstat-collect.service**

We can go even further and explore the content of the **sa1** script if we like

**cat /usr/lib64/sa/sa1**

The other pair is similar, except that it takes care for the summary

First, the timer

**cat /usr/lib/systemd/system/sysstat-summary.timer**

And then the service

**cat /usr/lib/systemd/system/sysstat-summary.service**

Finally, the script that is in use is a different one

**cat /usr/lib64/sa/sa2**

We can check the active timers and when they are expected to run with

**systemctl list-timers**

By now, there should be already a file of collected data. Let us check

**ls -al /var/log/sa/**

We can read a parameter from the captured data with

**sar -u -f /var/log/sa/sa<day-no>**

We can even specify a time period that we are interested in

**sar -u -s 01:50:00 -e 02:50:00 -f /var/log/sa/sa<day-no>**

If we omit the file, it will search in the today’s file

**sar -u -s 01:50:00 -e 02:50:00**

We can indeed run a few commands in an interactive manner to check some parameters

For example, to check the queue length (number of tasks waiting to run) and the load average, we can execute

**sar -q**

Should we want to see memory utilization statistics, we can execute

**sar -r**

We can ask for the **CPU** utilization and make for example three (3) snapshot with two (2) seconds delay

**sar -u 2 3**

### Nagios

Log on to **M1** with a regular user account

Let us install and test a monitoring solution

#### Installation

For this, we must execute the following

**sudo dnf install epel-release**

Then enable the **PowerTools** repository

**sudo dnf config-manager --set-enabled powertools**

And finally, install **nagios** plus the required packages

**sudo dnf install nagios nagios-common nagios-selinux nagios-plugins-all**

Enable and start both **httpd** and **nagios** services

**sudo systemctl enable --now httpd**

**sudo systemctl enable --now nagios**

Then adjust the firewall

**sudo firewall-cmd --add-service={http,https} --permanent**

**sudo firewall-cmd --reload**

And then open a browser tab on the host and navigate to http://<m1-ip-address>/nagios

You will be asked for user credentials which you currently do not have

Let us correct this

Return to the session to **M1** and execute

**sudo htpasswd /etc/nagios/passwd nagiosadmin**

Now, we have the required credentials, so let us return to the web browser and try again

We can explore different views or sections - **Map**, **Hosts**, **Host Groups**, **Services** and **Service Groups**

So far, we can see that there is a warning, which we can ignore

#### Explore Plugins

Let us check what plugins are there

**ls -al /usr/lib64/nagios/plugins/**

We can get help for them with

**/usr/lib64/nagios/plugins/check\_ping -h**

And use them

**/usr/lib64/nagios/plugins/check\_ping -H 8.8.8.8 -w 2,20% -c 5,50%**

We could experiment with others as well, but let us move on

#### Monitor Hosts

In order to add the other two hosts, we must create the appropriate configuration

It can be stored either in one file, two separate files, or added to an existing (part of the default setup) file

Let us create two new files

The first one, we will create with

**sudo vi /etc/nagios/objects/host-m2.cfg**

And enter the following

**define host {**

**use linux-server**

**host\_name m2.lsaa.lab**

**alias M2 VM**

**address <m2-ip>**

**}**

Save and close the file

The second one will be similar

**sudo vi /etc/nagios/objects/host-m3.cfg**

And will contain almost the same

**define host {**

**use linux-server**

**host\_name m3.lsaa.lab**

**alias M3 VM**

**address <m3-ip>**

**}**

Save and close the file

Now, we must include both files in the configuration of **nagios**

**sudo vi /etc/nagios/nagios.cfg**

Add the following two lines

**cfg\_file=/etc/nagios/objects/host-m2.cfg**

**cfg\_file=/etc/nagios/objects/host-m3.cfg**

Save and close the file

Now, we can check if there are any errors

**sudo nagios -v /etc/nagios/nagios.cfg**

And if there are not any, we can restart **nagios** service

**sudo systemctl restart nagios**

And return to the browser to check the result

We should see our two newly added hosts there

#### Monitor Services

Let us add web server to be monitored on **M2** and a database to be monitored on **M3**

Log on to **M2** with a regular user

Install **Apache** with

**sudo dnf install httpd**

Create a simple **index.html** file with

**echo '<h1>Hello from M2</h1>' | sudo tee /var/www/html/index.html**

Enable and start the service

**sudo systemctl enable --now httpd**

Adjust the firewall

**sudo firewall-cmd --add-service http --permanent**

**sudo firewall-cmd --reload**

Log on to **M3** with a regular user

Install **MariaDB** with

**sudo dnf install mariadb-server**

Enable and start the service

**sudo systemctl enable --now mariadb**

Do the initial configuration

**sudo mysql\_secure\_installation**

Create a user for **nagios** to connect to the database

**mysql -u root -p -e "CREATE USER 'nagios'@'<m1-ip>' IDENTIFIED BY 'Password1'; FLUSH PRIVILEGES;"**

Adjust the firewall

**sudo firewall-cmd --add-port 3306/tcp --permanent**

**sudo firewall-cmd --reload**

Return on **M1**

Let us first add a check for the web server running on **M2**

Open the **host-m2.cfg** file for editing

**sudo vi /etc/nagios/objects/host-m2.cfg**

And add the following set of lines

**define service {**

**use generic-service**

**host\_name m2.lsaa.lab**

**service\_description WEB**

**check\_command check\_http!$HOSTADDRESS$**

**}**

Save and close the file

Test **nagios** configuration

**sudo nagios -v /etc/nagios/nagios.cfg**

Restart the service

**sudo systemctl restart nagios**

Return to the browser and check again

Go back to the command line

Let us check if we have a plugin for **MySQL**

**ls -al /usr/lib64/nagios/plugins/\*mysql\***

Okay, we have. So, how we can use the **check\_mysql** plugin?

**/usr/lib64/nagios/plugins/check\_mysql -h**

So, we must be able to check the connectivity to our database host (**M3**) with

**/usr/lib64/nagios/plugins/check\_mysql -H <m3-ip> -u nagios -p Password1**

Now, it is time to define a command and a service

Open the file for **M3**

**sudo vi /etc/nagios/objects/host-m3.cfg**

And add the following at the end to define the command

**define command {**

**command\_name check\_mysql\_cmdlinecred**

**command\_line $USER1$/check\_mysql -H '$HOSTADDRESS$' -u '$ARG1$' -p '$ARG2$'**

**}**

And then add the following to define the service

**define service {**

**use generic-service**

**host\_name m3.lsaa.lab**

**service\_description DB**

**check\_command check\_mysql\_cmdlinecred!nagios!Password1**

**}**

Save and close the file

Test **nagios** configuration

**sudo nagios -v /etc/nagios/nagios.cfg**

Restart the service

**sudo systemctl restart nagios**

Return to the browser and check again

### Observium

Let us test another monitoring solution which uses a different approach

Reset the infrastructure or spin up a new set of machines

Log on to **M1**

#### Installation

Install the required repositories

**sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm**

**sudo dnf install http://yum.opennms.org/repofiles/opennms-repo-stable-rhel8.noarch.rpm**

**sudo dnf install http://rpms.remirepo.net/enterprise/remi-release-8.rpm**

**sudo dnf install yum-utils**

Enable the new **PHP** module

**sudo dnf module enable php:remi-7.4**

Install the required packages

**sudo dnf install tar wget httpd php php-opcache php-mysqlnd php-gd php-posix php-pear cronie net-snmp net-snmp-utils fping mariadb-server mariadb rrdtool subversion whois ipmitool graphviz ImageMagick php-sodium python3 python3-mysql python3-PyMySQL**

Change the default **Python** version

**sudo alternatives --set python /usr/bin/python3**

Create the target folders

**sudo mkdir -p /opt/observium/{rrd,logs}**

Change the ownership

**sudo chown apache:apache /opt/observium/{rrd,logs}**

Enter the **/opt** folder

**cd /opt**

Download the installation archive

**sudo wget https://www.observium.org/observium-community-latest.tar.gz**

Extract its content

**sudo tar zxvf observium-community-latest.tar.gz**

Enable and start the database

**sudo systemctl enable --now mariadb**

Run the initial configuration

**sudo mysql\_secure\_installation**

Create database

**mysql -u root -p -e "CREATE DATABASE observium DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;"**

And then a user

**mysql -u root -p -e "GRANT ALL PRIVILEGES ON observium.\* TO 'observium'@'localhost' IDENTIFIED BY 'observium'; FLUSH PRIVILEGES;"**

Create a configuration out of the sample one

**cp /opt/observium/config.php.default /opt/observium/config.php**

Change the credentials information for accessing the database

**sed -i 's/USERNAME/observium/g' /opt/observium/config.php**

**sed -i 's/PASSWORD/observium/g' /opt/observium/config.php**

Add information where the **fping** utility can be found

**echo "\$config['fping'] = '$(which fping)';" >> /opt/observium/config.php**

We are ready to initialize the database

**/opt/observium/discovery.php -u**

Create a **cron** schedule file

**sudo vi /etc/cron.d/observium**

With the following content

**# Run a complete discovery of all devices once every 6 hours**

**33 \*/6 \* \* \* root /opt/observium/discovery.php -h all >> /dev/null 2>&1**

**# Run automated discovery of newly added devices every 5 minutes**

**\*/5 \* \* \* \* root /opt/observium/discovery.php -h new >> /dev/null 2>&1**

**# Run multithreaded poller wrapper every 5 minutes**

**\*/5 \* \* \* \* root /opt/observium/poller-wrapper.py >> /dev/null 2>&1**

**# Run housekeeping script daily for syslog, eventlog and alert log**

**13 5 \* \* \* root /opt/observium/housekeeping.php -ysel**

**# Run housekeeping script daily for rrds, ports, orphaned entries in the database and performance data**

**47 4 \* \* \* root /opt/observium/housekeeping.php -yrptb**

Save and close the file

Change its permissions with

**sudo chmod --reference /etc/cron.d/0hourly /etc/cron.d/observium**

Reload the **cron** daemon

**sudo systemctl reload crond**

Create administrator user

**/opt/observium/adduser.php admin admin 10**

Create a virtual host configuration file

**sudo vi /etc/httpd/conf.d/observium.conf**

And enter the following

**<VirtualHost \*>**

**DocumentRoot /opt/observium/html/**

**ServerName m1.lsaa.lab**

**CustomLog /var/log/httpd/access\_log\_observium combined**

**ErrorLog /var/log/httpd/error\_log\_observium**

**<Directory "/opt/observium/html/">**

**AllowOverride All**

**Options FollowSymLinks MultiViews**

**Require all granted**

**</Directory>**

**</VirtualHost>**

Save and close the file

Change the context of the document folder

**chcon -Rv --type=httpd\_sys\_content\_t /opt/observium/html/**

Enable and start the **httpd** service

**sudo systemctl enable --now httpd**

Adjust the firewall

**sudo firewall-cmd --add-service={http,https} --permanent**

**sudo firewall-cmd --reload**

And then open a browser tab on the host and navigate to http://<m1-ip-address>

You will be asked for user credentials, use **admin** with password **admin**

#### Prepare the Clients

Log on to **M2**

Install the required packages

**sudo dnf install httpd net-snmp wget**

Download one additional component

**sudo wget -O /usr/local/bin/distro https://gitlab.com/observium/distroscript/raw/master/distro**

Set it to be executable

**sudo chmod +x /usr/local/bin/distro**

Copy the provided **snmpd.conf** file to the machine

*Don’t forget to change the* ***<m1-ip-address>*** *placeholder to match your setup*

And move it to **/etc/snmp/snmpd.conf**

Restore the **SELinux** context for the file

**sudo restorecon /etc/snmp/snmpd.conf**

Enable and start the **httpd** and **snmpd** services

**sudo systemctl enable --now httpd**

**sudo systemctl enable --now snmpd**

Adjust the firewall

**sudo firewall-cmd --add-service={http,https} --permanent**

**sudo firewall-cmd --add-port=161/udp --permanent**

**sudo firewall-cmd --reload**

Set a small **index.html** file

**echo "<h1>$HOSTNAME</h1>" | sudo tee /var/www/html/index.html**

Repeat the steps on **M3** as well

#### Add the Clients

Return on **M1**

Test the connectivity to **M2**

**fping m2**

And try to get information via **SNMP**

**snmpget -v2c -c public m2 sysObjectID.0**

Now go to the web interface and add **M2**

Go to **Devices** and then **Add Device**

Enter the host name and the community name (**public**)

And confirm with **Add device**

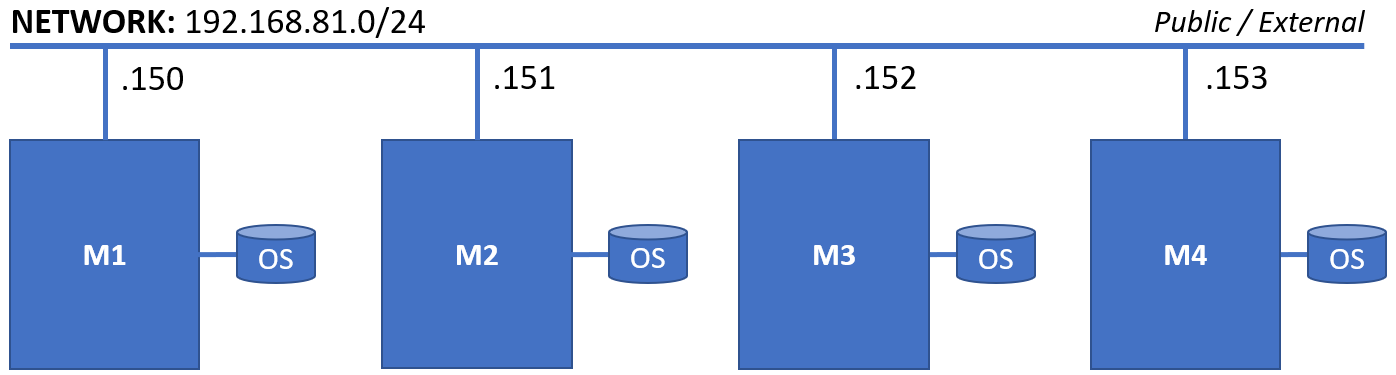
Return to the console on **M1** and add **M3**

**/opt/observium/add\_device.php m3**

Return to the web interface and explore

## Part 2: Configuration Management

For this part we will need an infrastructure like this one:



Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

### Ansible

Do not forget to reset the machines used during the previous part or spin a new set

Log on to the **M1** machine with a regular user account

This will be our control host

#### Installation

Install the **EPEL** repository

**sudo dnf install epel-release**

Now, install the **ansible** package

**sudo dnf install ansible**

Check that the tool is working

**ansible --version**

#### Inventory

Explore the sample inventory file

**cat /etc/ansible/hosts**

Should you want, create a copy of the file and open the original one for editing

**sudo vi /etc/ansible/hosts**

Enter the following

**[web]**

**m2.lsaa.lab**

**m3.lsaa.lab [db]**

**m4.lsaa.lab**

Save and close the file

#### Host Connectivity

Generate a key pair

**ssh-keygen**

Copy the public key to all machines

**ssh-copy-id user@m2.lsaa.lab**

**ssh-copy-id user@m3.lsaa.lab**

**ssh-copy-id user@m4.lsaa.lab**

Not that there is much sense but let us execute the following

**ansible m2.lsaa.lab -a hostname**

This way we got the host name (that we already knew) of the host

Of course, we can change the command to something else

In order to get the free space, we can execute

**ansible m2.lsaa.lab -m command -a "df -h"**

Or

**ansible m2.lsaa.lab -m shell -a "df -h"**

The result appears to be the same but in fact the **command** module is processed directly while the **shell** is passed through a **shell**

Furthermore, the **command** module can be omitted, as we did in the beginning

To see the difference between those two, let us execute

**ansible m2.lsaa.lab -m command -a 'echo $HOSTNAME'**

**ansible m2.lsaa.lab -m shell -a 'echo $HOSTNAME'**

We can address command to a group of machines from the inventory

**ansible web -m shell -a "free -m"**

Or to all machines in the inventory

**ansible all -m shell -a "free -m"**

#### Explore Modules

We can list the available modules with

**ansible-doc -l**

We can search, move forward or backward, etc.

Or ask directly for information for a module

**ansible-doc dnf**

Let us use the **dnf** module to install **httpd** on **M2**

The command should look like

**ansible m2.lsaa.lab -m dnf -a "name=httpd state=present" --become -K**

Explore the documentation for the **service** module

The command to start and enable the service should look like

**ansible m2.lsaa.lab -m service -a "name=httpd state=started enabled=true" --become -K**

And finally, we must understand how we can enable a service in the firewall

It should look similar to

**ansible m2.lsaa.lab -m firewalld -a "service=http state=enabled permanent=yes" --become -K**

Let us check by opening a browser tab on the host

It is not working

Hm, maybe we should find a way to alter the running configuration as well

Return to the documentation and explore further

The command should be changed to

**ansible m2.lsaa.lab -m firewalld -a "service=http state=enabled permanent=yes immediate=yes" --become -K**

Now, check again in the browser

It should work now

There is a module for files, so check in the documentation what it is doing

No, it creates only empty files

Maybe we can copy a file from our control station to a host

Let us explore the copy module

Now, create a simple index file

**echo "<h1>Powered by Ansible</h1>" > index.html**

The command to copy the file should look like

**ansible m2.lsaa.lab -m copy -a "src=index.html dest=/var/www/html/index.html owner=apache group=apache" --become -K**

Should we want to see detailed information about what is going on, we can add **-v** or **-vv** and so on to the end of the command

Now, if we check again in the browser, we will see our custom index page

#### Play

It would be nice, if we can put all those commands into a script and run them against all web servers for example

Create a new file and open it for editing

**vi play.yaml**

Enter the following for a start

**---**

**- hosts: web**

**become: true**

You should be careful with the indent (use spaces, not tabs)

Then add a task to install the web server

**tasks:**

**- name: Install Apache HTTP Server**

**dnf: name=httpd state=present**

Next, add a task (but skip the word **tasks**) to start the service

**- name: Start Apache HTTP Server and Enable it**

**service: name=httpd state=started enabled=true**

As a next step, we opened a service in the firewall, so we must repeat it here

**- name: Allow HTTP service in the firewall**

**firewalld: service=http state=enabled permanent=yes immediate=yes**

And as a final step, we must copy the custom **index.html** file

For this one we will use alternative (***dictionary based***) syntax

**- name: Copy custom index.html file**

**copy:**

**src: index.html**

**dest: /var/www/html/index.html**

**owner: apache**

**group: apache**

Of course, in one file or project, we should stick to one syntax type (whichever we prefer)

Save and close the file

Check its syntax with

**ansible-playbook play.yaml --syntax-check**

If there are any errors, go and correct them

We can check which hosts will be affected with

**ansible-playbook play.yaml --list-hosts**

And finally, we can execute the script with

**ansible-playbook play.yaml**

Hm, it did not succeed. Why? We missed one very important part. Try with this one

**ansible-playbook play.yaml -K**

If we open a browser tab on the host and check **M2** and **M3** both should have a reachable custom index web page

#### Playbook

We can combine multiple plays in a playbook

For example, we can extend our play, and add instructions for database installation on the **db** group

Open the file for editing

**vi play.yaml**

Go to the end of the file and switch to editing mode

Enter the following

**- hosts: db**

**become: true**

Then the installation

**tasks:**

**- name: Install MariaDB Server**

**dnf: name=mariadb-server state=present**

And finally, the service part

**- name: Start MariaDB Server and Enable it**

**service: name=mariadb state=started enabled=true**

Save and close the file

Test it with for correctness

**ansible-playbook play.yaml --syntax-check**

Check the affected hosts

**ansible-playbook play.yaml --list-hosts**

Execute it with

**ansible-playbook play.yaml -K**

## Part 3: Backup and Restore

For this part we will need an infrastructure like this one:

Chart, waterfall chart

Description automatically generated

Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

Do not forget to reset the machines used during the previous part or spin a new set

### Conventional Tools

Log on to the **M1** machine

#### tar

Make sure that all required packages are there

**sudo dnf install tar bzip2 gzip xz**

Now, let us create a **tar** archive

**sudo tar -cvf etc.tar /etc**

And one with **gzip** compression

**sudo tar -cvzf etc.tar.gz /etc**

Second one with **bzip** compression

**sudo tar -cvjf etc.tar.bz2 /etc**

And third one with **xz** compression

**sudo tar -cvJf etc.tar.xz /etc**

If we compare them, we will notice a difference in their size

**ls -lhS**

Of course, there was difference in the execution time as well

When we do want to list the content of an archive, we can use

**tar -tvf etc.tar**

For a compressed archive, we must add the appropriate switch. For example, for **gzip** the command will become

**tar -tzvf etc.tar.gz**

In order to extract archive content, we must change the **-t** switch to **-x** switch

#### dd

While still on the **M1** machine, make sure that there is a DVD attached to it (for example, a small ISO like the NET install image)

There is no need to be mounted. We can check with

**lsblk**

Should we want to make image of just a small part (for example first 10 MB) of a media, we can execute

**sudo dd if=/dev/sr0 of=cdrom-part.iso bs=1M count=10**

Of course, we can achieve the same by changing the values of **bs** (block size) and **count** (count of blocks)

**sudo dd if=/dev/sr0 of=cdrom-part-1.iso bs=10M count=1**

If we want to make a copy of the whole media, we can omit the block size and block count parts

**sudo dd if=/dev/sr0 of=cdrom.iso**

We can add progress information as well

**sudo dd if=/dev/sr0 of=cdrom-1.iso status=progress**

In case we need to restore the information, we can use the **iso** file as **input file** and a target device as **output file**

This image, we can mount and use as if it is a regular device

**sudo mount -o ro,loop cdrom.iso /mnt**

And interact with its content

**ls -alR /mnt**

### Rsync

Prepare the playground

Install the required packages

**sudo dnf install rsync rsync-daemon**

Create a folder for the backup

**mkdir backup**

#### Local Copy

Backup or mirror a folder here. For example, this can be **/usr/share/doc**

**rsync -arv /usr/share/doc ~/backup/**

This will copy the folder and its sub-folders

We can press **Ctrl+C** at any time to stop the operation

Should we run the same command again, the operation will continue from where it was interrupted

**rsync -arv /usr/share/doc ~/backup/**

Let us check the result

**ls -alR ~/backup/doc**

And remove the backed-up **doc** folder

**rm -rf ~/backup/doc**

If we want to copy only the folder content, we must change the command to

**rsync -arv /usr/share/doc/ ~/backup/**

#### Remote Copy

Switch to **M2**

Install the **rsync** package

**sudo dnf install rsync**

Create a folder to contain the **backup**

**mkdir backup**

Initiate a synchronization from the remote host (**M1**) to the backup folder here (**M2**) via **ssh**

**rsync -rve ssh user@m1.lsaa.lab:backup/ ~/backup/**

#### Rsync Server

Return to **M1** machine

This will become our target machine for storing archived content

Create a **/home/backup** folder

**sudo mkdir /home/backup**

Let us modify the configuration

**sudo vi /etc/rsyncd.conf**

Using the sample configuration, add the following block (or uncomment the appropriate lines) at the end

**max connections = 4**

**pid file = /var/run/rsyncd.pid**

**log file = /var/log/rsyncd.log**

**transfer logging = yes**

Next, add the endpoint block

**[backup]**

**path = /home/backup**

**hosts allow = <m2-ip-address>**

**hosts deny = \***

**list = true**

**uid = root**

**gid = root**

**read only = false**

Save and close the file

Enable and start the service

**sudo systemctl enable --now rsyncd**

Adjust the **SELinux** policy and set **rsyncd** to be in permissive mode

**sudo semanage permissive -a rsync\_t**

*If the command is not found, install it with* ***sudo dnf install policycoreutils-python-utils***

Then adjust the firewall rules

**sudo firewall-cmd --add-service=rsyncd --permanent**

**sudo firewall-cmd --reload**

Return to **M2** which will be our source in this case

Execute the following to backup all files from **~/backup/** folder using compression to the **M1** machine’s **backup** endpoint

**rsync -avz ~/backup/ m1.lsaa.lab::backup**

Execute the following to backup all files except the **README** like files from **~/backup/** folder using compression to the **M1** machine’s **backup** endpoint and delete them from the target

**rsync -avz --delete-excluded --exclude='README\*' ~/backup/ m1.lsaa.lab::backup**

### Bacula

For this part we will need just one machine

We will install all components on **M1**

#### Installation

Log on to the **M1** machine

Install the required packages

**sudo dnf install bacula-director bacula-storage bacula-console bacula-client mariadb-server**

Enable and start the **mariadb** service

**sudo systemctl enable --now mariadb**

Do the initial configuration

**sudo mysql\_secure\_installation**

Change the default database engine (from PostgreSQL to MySQL/MariaDB)

**sudo alternatives --config libbaccats.so**

Create a database for the **bacula** solution

**mysql -u root -p -e "CREATE DATABASE bacula DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;"**

And then a user

**mysql -u root -p -e "GRANT ALL PRIVILEGES ON bacula.\* TO 'bacula'@'%' IDENTIFIED BY 'bacula'; FLUSH PRIVILEGES;"**

Create the tables

**sudo /usr/libexec/bacula/make\_mysql\_tables -p**

Adjust the **SELinux** policy to permissive for **bacula**

**sudo semanage permissive -a bacula\_t**

*If the command is not found, install it with* ***sudo dnf install policycoreutils-python-utils***

#### Director configuration

Open the director configuration for edit

**sudo vi /etc/bacula/bacula-dir.conf**

On row **25** set the console password you want to use (for example **DirectorPassword1**)

Change row **125** to point to **/home**

This way, we set up what to backup

Set password on row **177** to something you want (for example **FilePassword1**)

On row **203** set the **address** to the **FQDN** of the machine - **m1.lsaa.lab**

On row **205** set the password to something (for example **StoragePassword1**)

Set the **dbpassword** on row **242** to **bacula**

Comment out the last few lines (319+) related to the tray monitor process

Save and close the file

Open the console configuration file

**sudo vi /etc/bacula/bconsole.conf**

And change the password on row **12** to **DirectorPassword1** or the one you specified earlier

Save and close the file

Enable and start the **bacula director**

**sudo systemctl enable --now bacula-dir**

#### Storage Configuration

Open the storage configuration file

**sudo vi /etc/bacula/bacula-sd.conf**

Change the password on row **31** to **StoragePassword1** or the one used earlier

Comment out the next director section

Save and close the file

Enable and start the service

**sudo systemctl enable --now bacula-sd**

#### File Configuration

Open the configuration file for editing

**sudo vi /etc/bacula/bacula-fd.conf**

Modify the password on row **19** to **FilePassword1** or the one you set earlier

Comment out the director section for monitoring service

Save and close the file

Enable and start the service

**sudo systemctl enable --now bacula-fd**

#### Backup

Let us create a backup of our **/home** directory

Start the console

**sudo bconsole**

Enter **label** to list the storage resources

Select the first by entering **1** and pressing **Enter**

For autochanger select the default value by hitting **Enter**

Enter name for the volume. For example, **V20211015**

Press **Enter** key again to confirm the default slot (0)

Enter **2** to select the **File** pool

Execute **run** to list the available jobs

Select **1** and press **Enter** to run the first one

Confirm with **yes** and then **Enter**

Enter the **messages** command to see the list of generated messages

Execute it again after 20 seconds to see the completion information

Type **exit** to exit the console

Check where the archive is stored

**ls -al /tmp**

Check the file type with

**sudo file /tmp/V20211015**

#### Restore

Start the console again

**sudo bconsole**

Type **restore** and hit **Enter**

Select **1** to see the list of the last 20 jobs

Then enter **5**

Execute **ls** to see the content of the archive

To mark content for restoration, execute **mark home**

Then check its content **lsmark**

Then execute **done** to submit the content for restoration

Confirm with **yes**

Then check the status with **messages**

Close the console with **exit**

List the restored location

**ls -al /tmp/bacula-restores/home/**