Experiment 1

Aim: To demonstrate the creation of a Virtual Machine.

Description:

Virtual Machine: A Virtual machine, often abbreviated as VM is an entire computer system that is run via emulation.

Hypervisor: A Hypervisor is a program that facilitates the creation and handling of Virtual Machines. It is the task of a hypervisor to allocate resources to its VMs in the form of VRAM, VCPUs and Virtual storage.

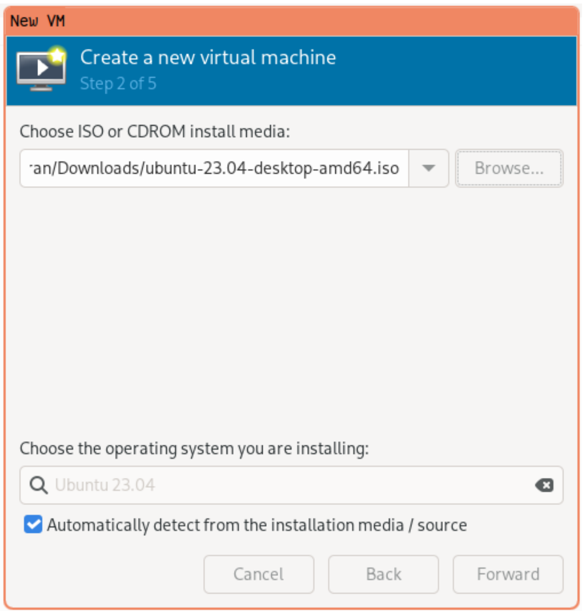
In this experiment, we will be using QEMU, a libre and open source hypervisor that leverages the KVM capabilities on a GNU/Linux system, allowing for high performance and efficient runs.

Procedure and Outputs:

1. Install Virt Manager on your GNU/Linux system.

```
sudo apt install virt-manager
```

2. Download the installation image for the guest operating system of your choice.
3. Start virt-manager and create a new virtual machine.
4. Select the Local install media option and provide the install image.
5. Select the CPU and RAM to be allocated to the VM, followed by the storage.

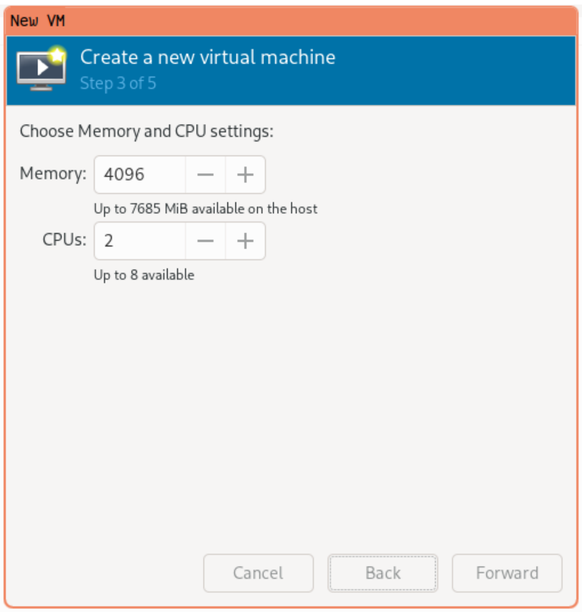


Create a new virtual machine
Step 2 of 5

Choose ISO or CDROM install media:
'an/Downloads/ubuntu-23.04-desktop-amd64.iso

Choose the operating system you are installing:
Ubuntu 23.04

☑ Automatically detect from the installation media / source



Create a new virtual machine
Step 3 of 5

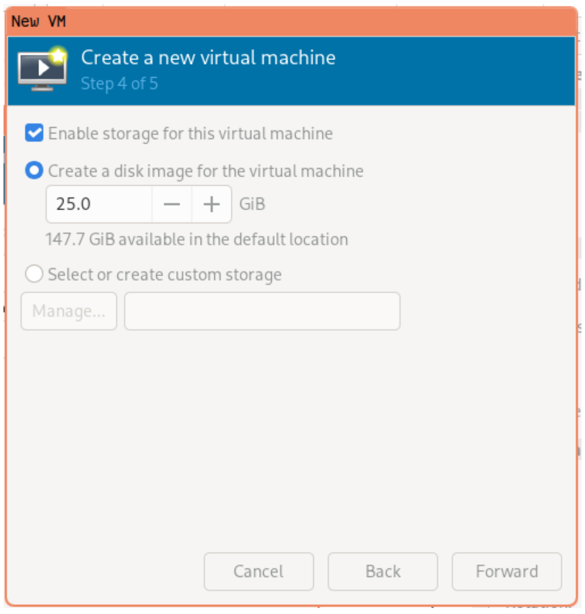
Choose Memory and CPU settings:

Memory: 4096
Up to 7685 MiB available on the host

CPUs: 2
Up to 8 available

6. Finish the install by confirming the provided options.

7. Go through the installation process of your guest OS.



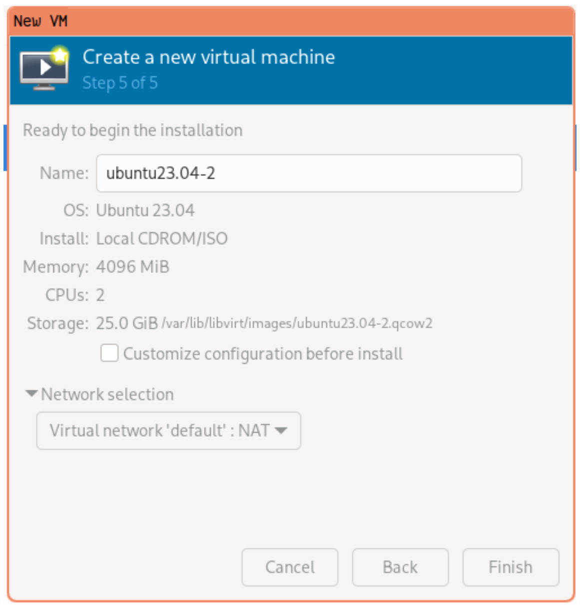
[✓] Enable storage for this virtual machine

[● Create a disk image for the virtual machine]

25.0 GiB

147.7 GiB available in the default location

○ Select or create custom storage



Ready to begin the installation

Name: ubuntu23.04-2

OS: Ubuntu 23.04

Install: Local CDROM/ISO

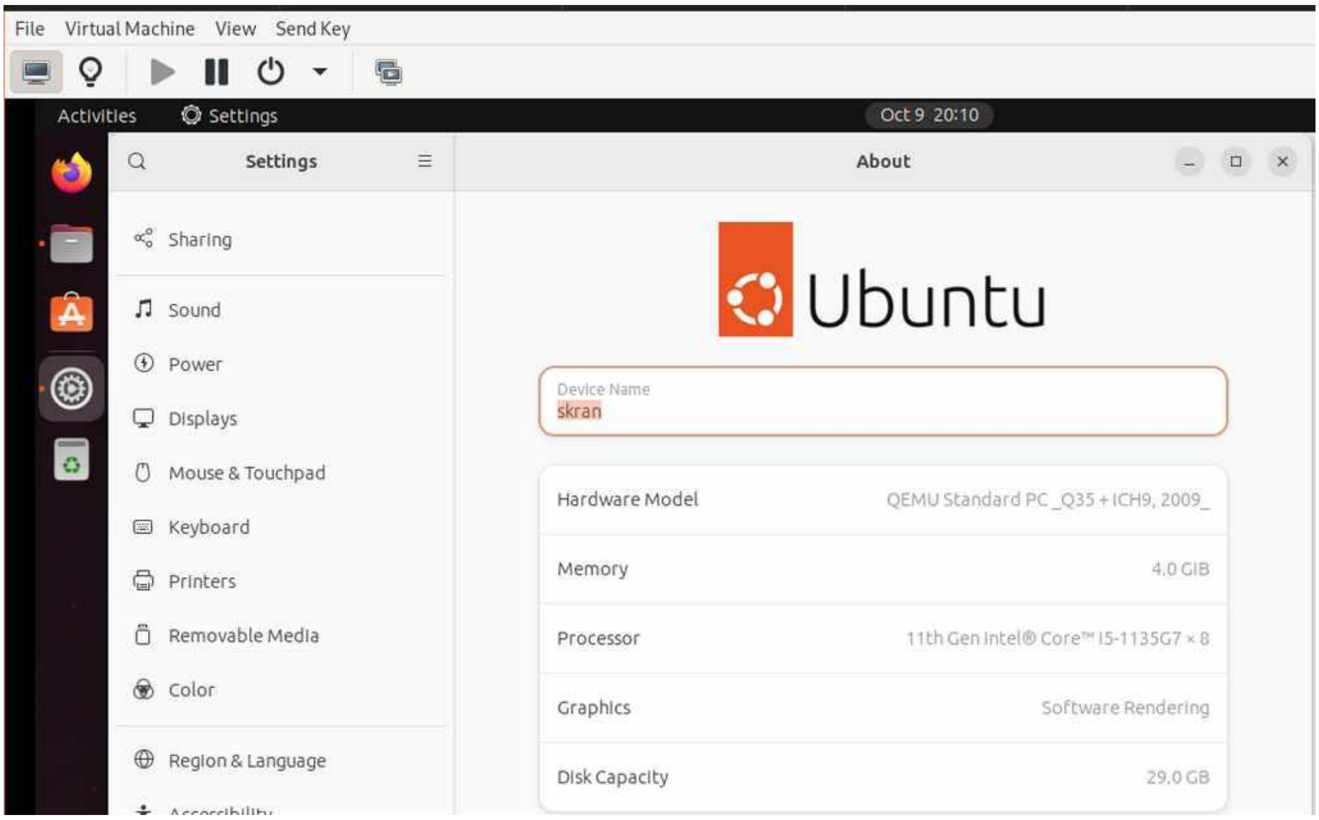
Memory: 4096 MiB

CPUs: 2

Storage: 25.0 GiB /var/lib/libvirt/images/ubuntu23.04-2.qcow2

☐ Customize configuration before install

Virtual network 'default' : NAT



Device Name
skran

Hardware Model	QEMU Standard PC _Q35 + ICH9, 2009_
Memory	4.0 GiB
Processor	11th Gen Intel® Core™ i5-1135G7 × 8
Graphics	Software Rendering
Disk Capacity	29.0 GB

Conclusions:

A Virtual installation of the Ubuntu Operating system has successfully been created.

The configurations used were:

- 4 GiB of RAM
- 2 vCPUs (2 threads of main CPU)
- 25 GiB of storage.

160120749051

Experiment 2

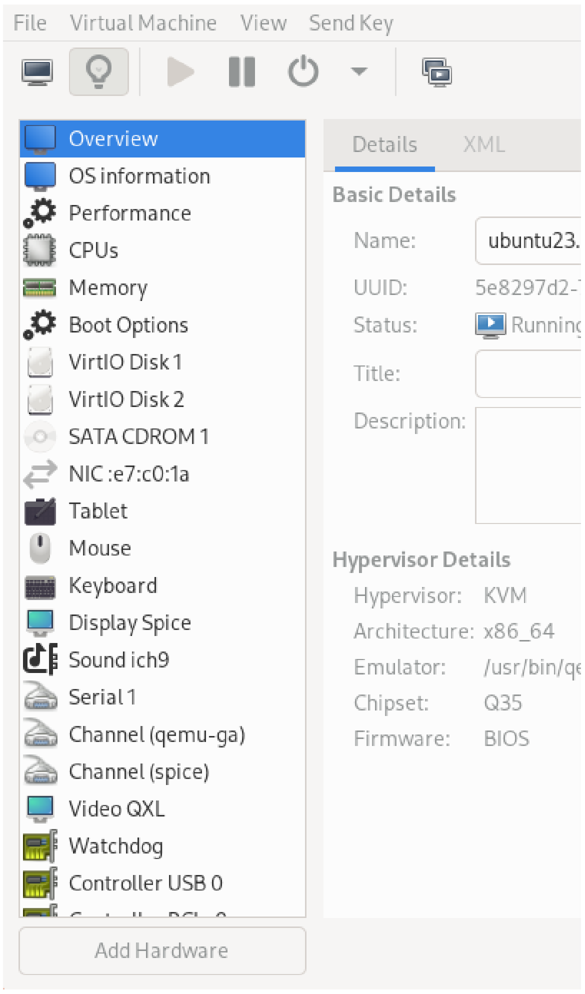
Aim: To demonstrate the addition of a Virtual block to a Virtual Machine.

Description:

Virtual Block: A Virtual Block refers to a block of storage that is allocated to the VM by the hypervisor.

Procedure and Outputs:

1. Open the configuration menu of an existing Virtual Machine and click on “Add Hardware”.
2. Select the “Storage” section and enter the desired size of the storage block to be added.
3. Click on finish and start the VM.
4. Verify the addition of the new block from the Guest OS.

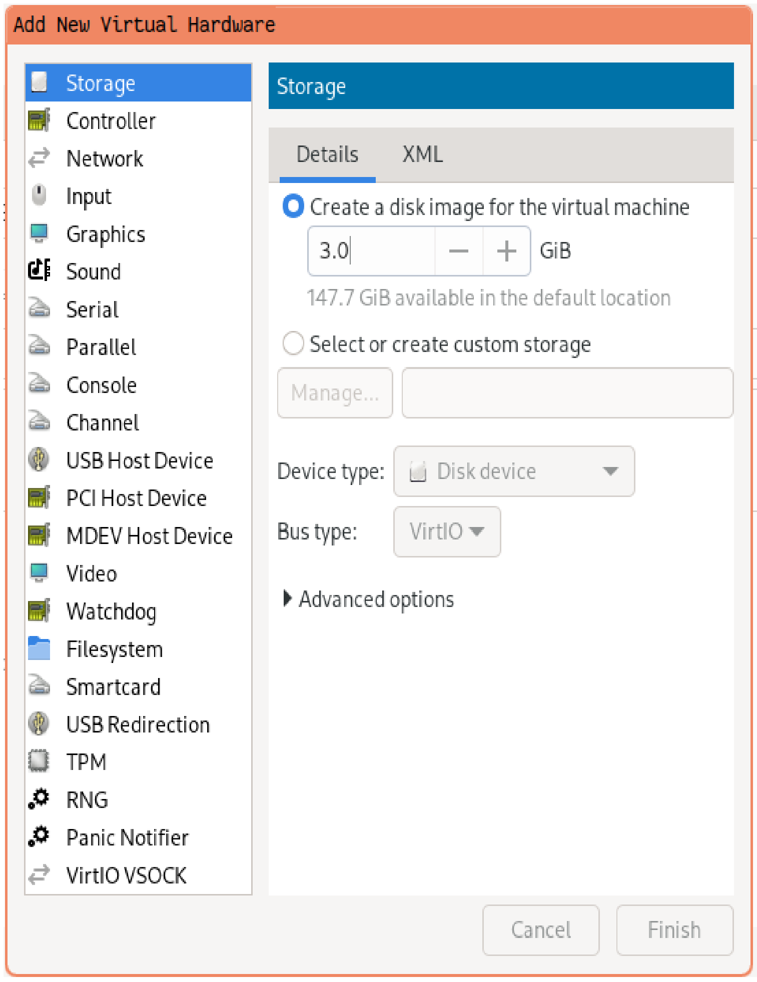


Basic Details

Name: ubuntu23.
UUID: 5e8297d2-
Status: Running
Title:
Description:

Hypervisor Details

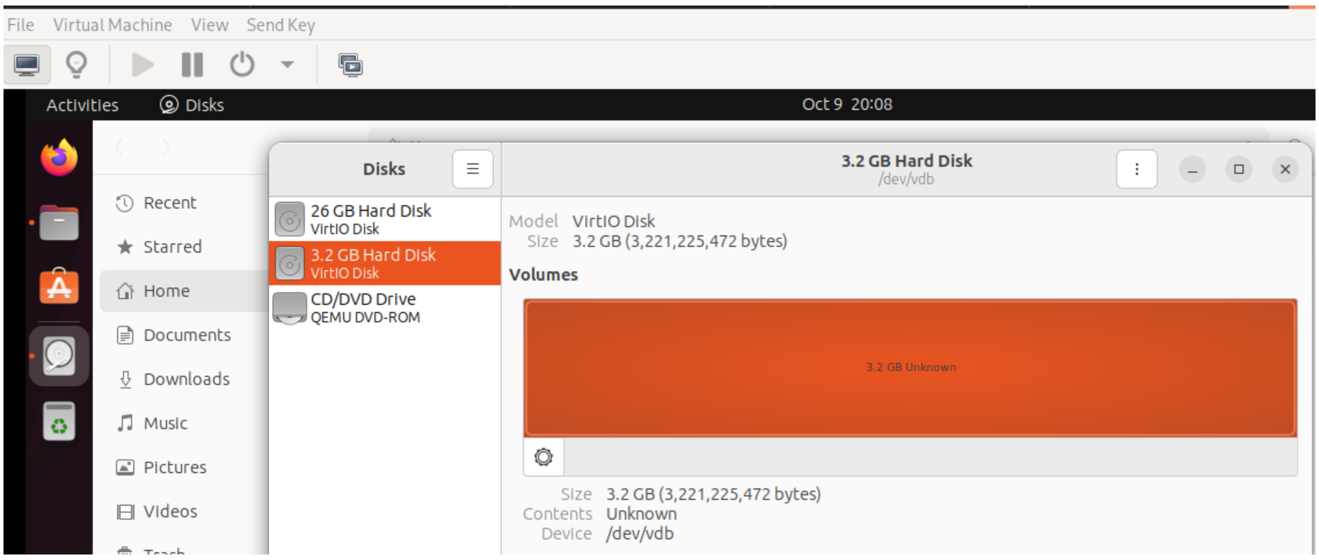
Hypervisor: KVM
Architecture: x86_64
Emulator: /usr/bin/qe
Chipset: Q35
Firmware: BIOS



Storage

[✓ Create a disk image for the virtual machine] 3.0 GiB
147.7 GiB available in the default location
Select or create custom storage

Device type: Disk device
Bus type: VirtIO



Model VirtIO Disk
Size 3.2 GB (3,221,225,472 bytes)

Size 3.2 GB (3,221,225,472 bytes)
Contents Unknown
Device /dev/vdb

Conclusions:

A Virtual Block of 3 GB has successfully been allocated to a Virtual Machine running the Ubuntu Operating system.

The block is detected by the Guest OS as shown in the images.

Addition of Virtual blocks is useful in scenarios such as:

- Increasing the storage of an existing Virtual Machine.
- Setting up a shared storage block among multiple Virtual Machines.

Experiment 3

Aim: To demonstrate the execution of a C program in a Virtual Machine.

Procedure and Outputs:

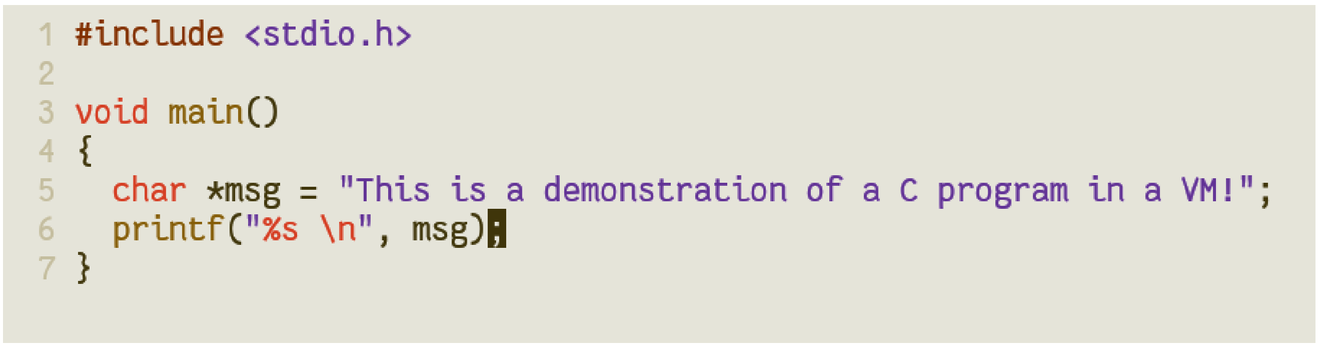
1. Install the gcc compiler for the Guest Operating system.

```
sudo apt install gcc
```

2. Open a text editor in the Virtual Machine and write a sample C program.

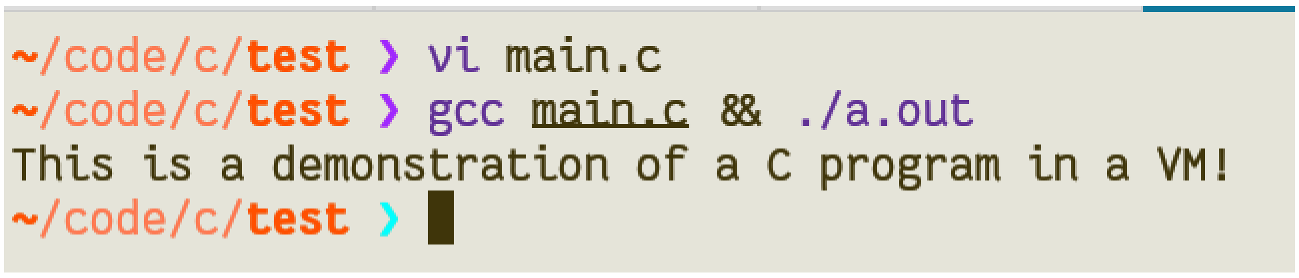
3. Compile and execute the C program using the command:

```
gcc program.c && ./a.out
```



```c
#include <stdio.h>

void main()
{
  char *msg = "This is a demonstration of a C program in a VM!";
  printf("%s \n", msg);
}
```



```
~/code/c/test ❯ vi main.c
~/code/c/test ❯ gcc main.c && ./a.out
This is a demonstration of a C program in a VM!
~/code/c/test ❯
```

Conclusions:

The C program successfully runs on a Guest Virtual machine running Arch Linux.

The vim editor has been used to write the program, which prints a string upon execution.

Experiment 4

Aim: To demonstrate the process of Virtual Machine Migration (cold).

Description:

Virtual Machine Migration: The migration of a Virtual Machine involves the transfer of all data from one hypervisor to another hypervisor, which may not necessarily be on the same hardware.

A cold migration occurs when the VM is powered off.

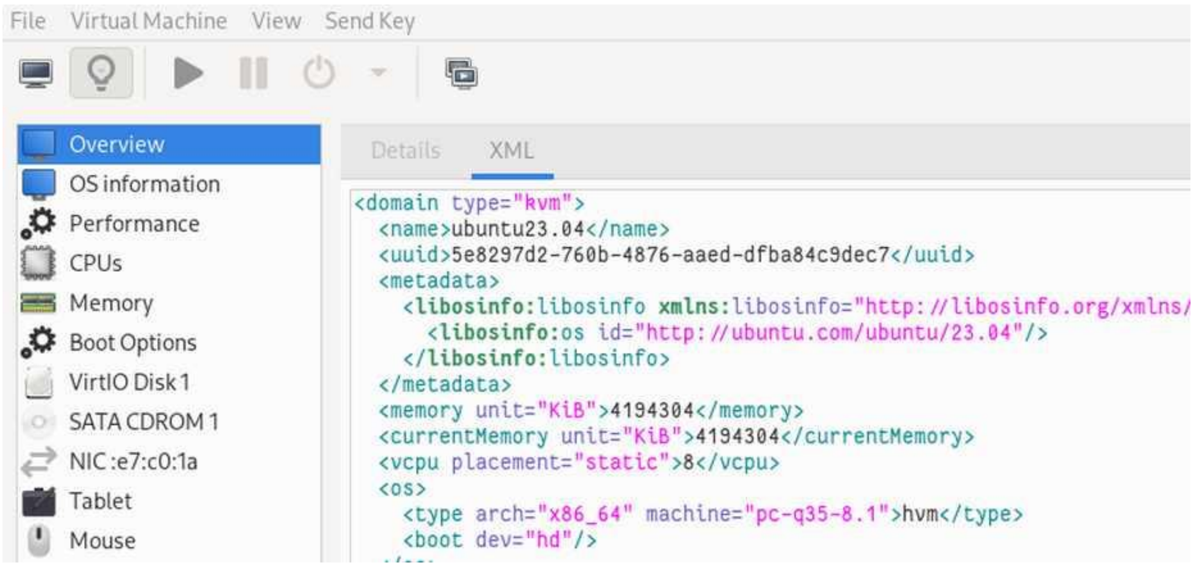
Procedure and Outputs:

1. Power off the Virtual Machine.

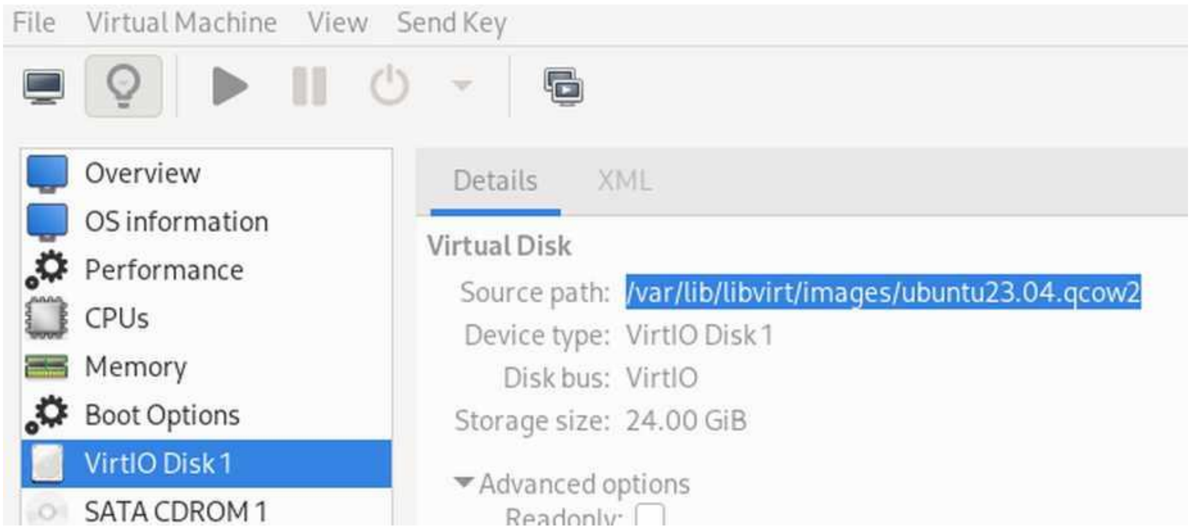
2. Go to the VM Configuration menu.

3. Store the XML contents in a file and transfer it to the destination machine.

4. Copy the qcow2 virtual disk image from the path mentioned under the disk section to the destination machine.



```
<domain type="kvm">
  <name>ubuntu23.04</name>
  <uuid>5e8297d2-760b-4876-aaed-dfba84c9dec7</uuid>
  <metadata>
    <libosinfo:libosinfo xmlns:libosinfo="http://libosinfo.org/xmlns/
      <libosinfo:os id="http://ubuntu.com/ubuntu/23.04"/>
    </libosinfo:libosinfo>
  </metadata>
  <memory unit="KiB">4194304</memory>
  <currentMemory unit="KiB">4194304</currentMemory>
  <vcpu placement="static">8</vcpu>
  <os>
    <type arch="x86_64" machine="pc-q35-8.1">hvm</type>
    <boot dev="hd"/>
```

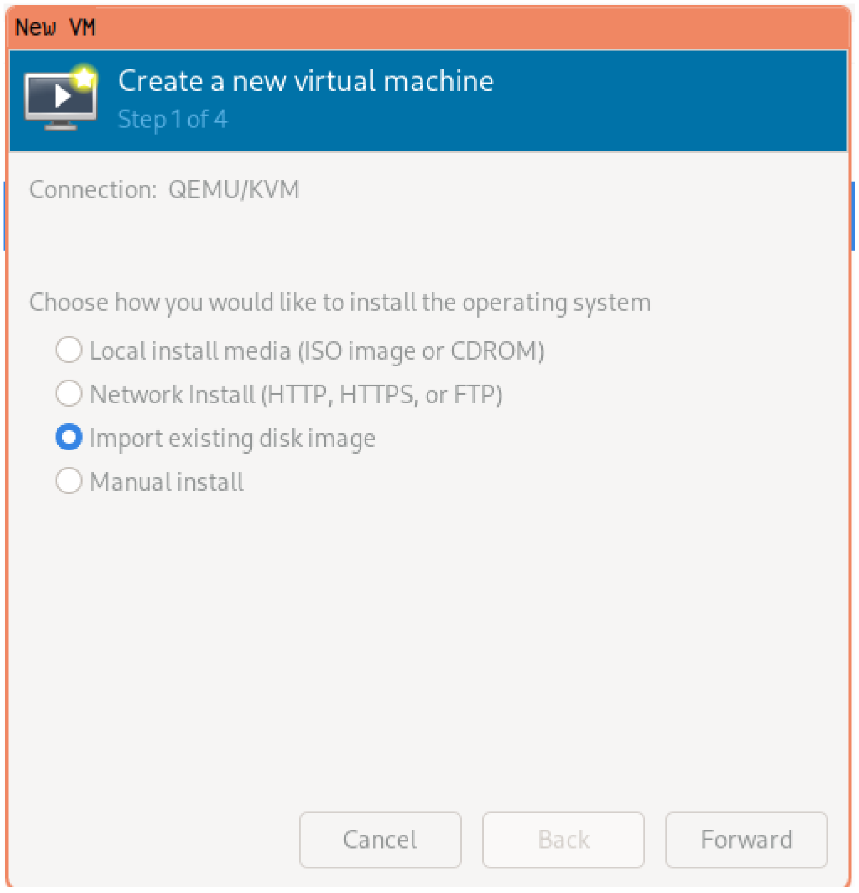


Virtual Disk

Source path: /var/lib/libvirt/images/ubuntu23.04.qcow2
Device type: VirtIO Disk 1
Disk bus: VirtIO
Storage size: 24.00 GiB

Advanced options
Readonly: ☐

5. Create a new Virtual machine on the destination hypervisor, specifying the transferred disk image.
6. Paste the XML configuration from the transferred file in the configuration section of the new VM.
7. Start the new VM and verify the migration.



Connection: QEMU/KVM

Choose how you would like to install the operating system

- Local install media (ISO image or CDROM)
- Network Install (HTTP, HTTPS, or FTP)
- [✓ Import existing disk image]
- Manual install

Conclusions:

Thus the cold migration of an Ubuntu VM has been successfully performed.

160120749051

Experiment 5

Aim: To demonstrate the installation of Apache Hadoop.

Description: The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.

It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

Hadoop is available only for GNU/Linux systems.

Procedure and Outputs:

1. Install the dependencies for hadoop. (Java Runtime v11, ssh and pdsh)

```
sudo apt install openjdk-11-jdk ssh
```

2. Download the latest stable release of hadoop from the official site.

3. Extract the tar archive into a dedicated directory.

4. Provide the directory of your java installation in

```
${hadoop_extracted_folder}/etc/hadoop/hadoop-env.sh
```

The path of your java installation is located in `/usr/lib/jvm/java-${version}-openjdk-amd64`

Edit the following line by placing the above mentioned path. For example,

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

5. Run hadoop using:

```
${hadoop_extracted_folder}/bin/hadoop
```

which should produce the following output upon successful installation

```
skran@skran:~/progs/hadoop-3.3.6$ bin/hadoop
Usage: hadoop [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]
 or    hadoop [OPTIONS] CLASSNAME [CLASSNAME OPTIONS]
  where CLASSNAME is a user-provided Java class

  OPTIONS is none or any of:

buildpaths                       attempt to add class files from build tree
--config dir                     Hadoop config directory
--debug                          turn on shell script debug mode
--help                           usage information
hostnames list[,of,host,names]   hosts to use in worker mode
hosts filename                   list of hosts to use in worker mode
loglevel level                   set the log4j level for this command
workers                          turn on worker mode
```

Conclusion:

Hadoop installation has been successfully performed on a system running Ubuntu.

160120749051

Experiment 6

Aim: To demonstrate the installation and running of Docker.

Description:

Containers: A container is a software unit that is bundled with all of its required dependencies and libraries and is run as an isolated process on a system.

- Containers allow for uniformity and consistency of runtimes across various operating systems and hardware via separation and isolation.
- They also allow for better security due to their controlled and isolated environment.
- In contrast to Virtual Machines, containers do not bundle entire operating systems and thus are much more lighter.

Docker: Docker is a popular libre and open source containerization software. It is used to create and manage programs as containers.

Procedure and Outputs:

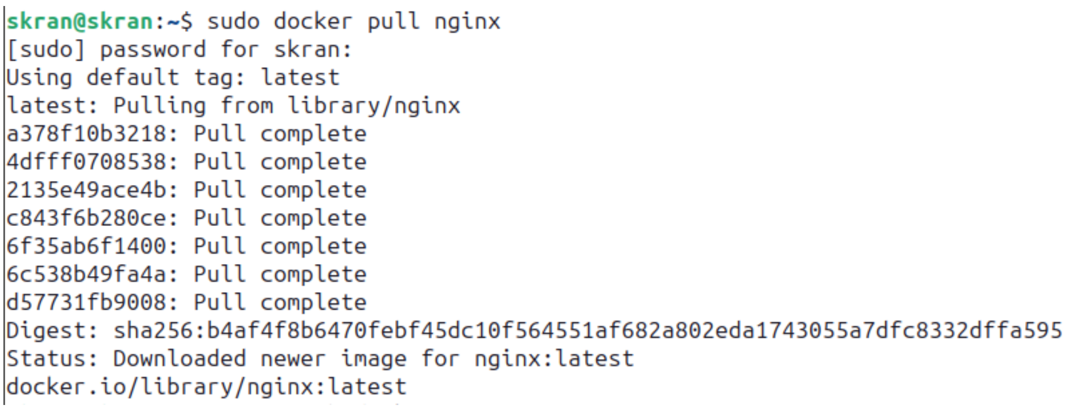
1. Install docker for your Operating System.

On Ubuntu: sudo snap install docker

2. Browse through available images on the docker hub website.

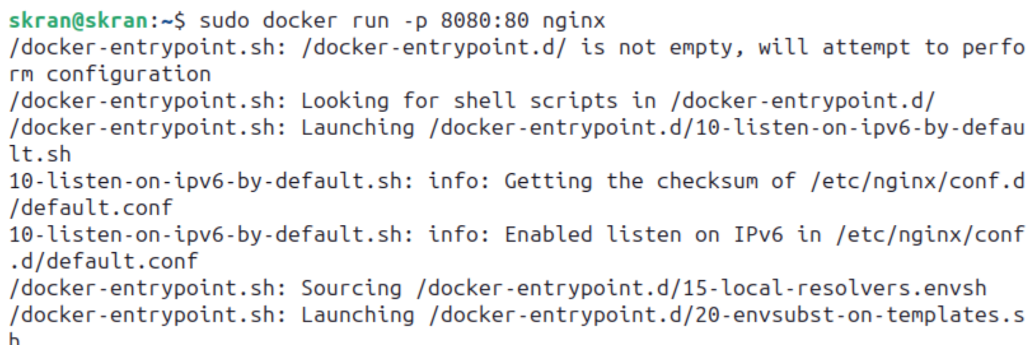
3. To install an Nginx container, run the following command:

```
sudo docker pull nginx
```



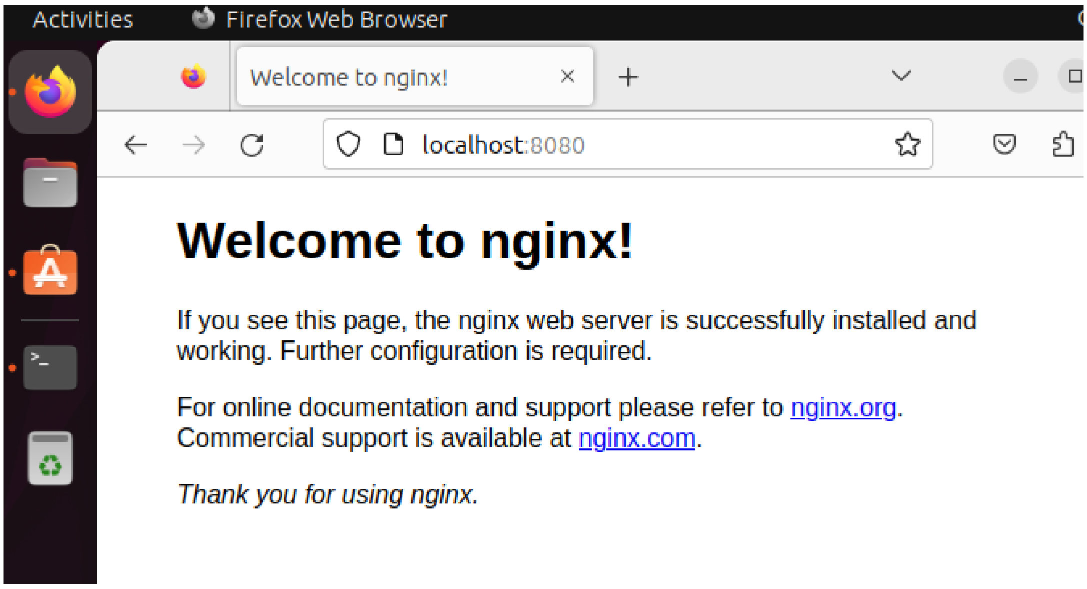
```
skran@skran:~$ sudo docker pull nginx
[sudo] password for skran:
Using default tag: latest
latest: Pulling from library/nginx
a378f10b3218: Pull complete
4dfff0708538: Pull complete
2135e49ace4b: Pull complete
c843f6b280ce: Pull complete
6f35ab6f1400: Pull complete
6c538b49fa4a: Pull complete
d57731fb9008: Pull complete
Digest: sha256:b4af4f8b6470febf45dc10f564551af682a802eda1743055a7dfc8332dffa595
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

4. Run the nginx image using the "docker run" command.



```
skran@skran:~$ sudo docker run -p 8080:80 nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perfo
rm configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-defau
lt.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d
/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf
.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.s
h
```

5. Verify the application launch (by visiting http://localhost:8080 in this case)



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

6. Use the following command to query the currently running containers.

sudo docker ps

```
skran@skran:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
 PORTS                                   NAMES
2072bf101af1   nginx     "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes
 0.0.0.0:8080->80/tcp, :::8080->80/tcp   loving_ganguly
```

7. Stop the container program using "docker stop "followed by a few characters of the container ID from the output of "docker ps":

sudo docker stop 2072b

Conclusion:

Thus we have successfully deployed a containerized version of the nginx web server using docker.

The container is fast and lightweight.

It is to be noted that while a console was used in this experiment, Docker also provides a graphical application called "Docker Desktop" that can be used to perform all the above.

Experiment 7

Aim: To demonstrate the setup of a single-node cluster in Hadoop.

Description: The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.

While Hadoop is supposed to be deployed across multiple nodes, this demonstration is performed on a single node for educational purposes.

Hadoop consists of two **NameNodes** (Primary and Secondary) followed by one or optionally more **DataNodes**.

Procedure and Outputs:

1. Install Hadoop (Refer exp 5).

2. Open the directory of your hadoop installation within your file manager application.
Note: The paths mentioned below will be with respect to this location.

3. Configure the NameNode by adding the below lines to the following file:

etc/hadoop/core-site.xml

```
<configuration>
 <property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
 </property>
</configuration>
```

4. Configure the DataNode by adding the below lines to the following file:

etc/hadoop/hdfs-site.xml

```
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
 </property>
</configuration>
```

5. The hadoop daemons communicate over SSH. Thus we set up passwordless SSH for the daemons. Use the command **ssh-keygen** and leave all prompts empty by pressing enter.

```
skran@skran:~/progs/hadoop-3.3.6$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/skran/.ssh/id_rsa):
Created directory '/home/skran/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/skran/.ssh/id_rsa
The key fingerprint is:
SHA256:8is0kpsxlBFsr+hvNhOe9FTvxSpmyfGM2FV/eXUKLE8 skran@skran
The key's randomart image is:
+---[RSA 3072]----+
+----[SHA256]-----+
```

6. Copy the SSH public key of your own machine into the authorized_keys file so that the daemons can use SSH seamlessly without a password.

```
skran@skran:~/$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

7. Format the HDFS filesystem.

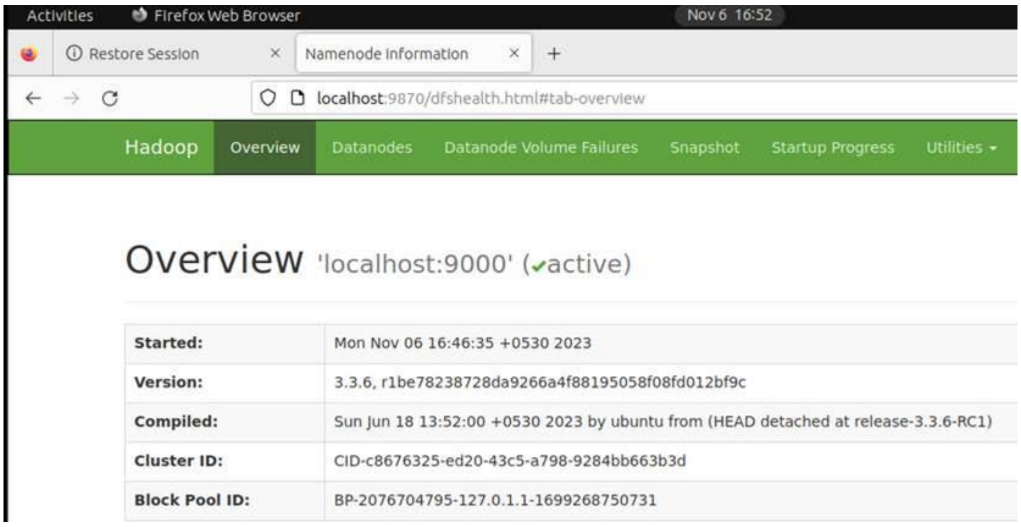
```
skran@skran:~/progs/hadoop-3.3.6$ bin/hdfs namenode -format
/************************************************************
STARTUP_MSG: Starting NameNode
2023-11-06 16:35:50,944 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when
meet shutdown.
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at skran/127.0.1.1
************************************************************/
```

8. Start the hadoop daemons by running the following script:

Note: This step does not work if pdsh is installed. If there is an error, run "sudo apt remove pdsh"

```
skran@skran:~/progs/hadoop-3.3.6$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [skran]
skran: Warning: Permanently added 'skran' (ED25519) to the list of known hosts.
```

9. Verify by visiting the Hadoop web interface at **http://localhost:9870**.



Overview 'localhost:9000' (✓active)

Started:	Mon Nov 06 16:46:35 +0530 2023
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 13:52:00 +0530 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-c8676325-ed20-43c5-a798-9284bb663b3d
Block Pool ID:	BP-2076704795-127.0.1.1-1699268750731

10. The daemons can be stopped by the following script:

```
skran@skran:~/progs/hadoop-3.3.6$ sbin/stop-dfs.sh
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [skran]
```

Conclusion:

The single node cluster has been successfully set-up.
It has been noticed that pdsh conflicts with the start-dfs.sh script and thus needs to be uninstalled.

References:

https://dev.to/donaldsebleung/setting-up-a-single-node-hadoop-cluster-p0a
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

Experiment 8

Aim: Demonstration of Unix commands in a Virtual Machine.

Description:

Unix: Unix is a family of multi-tasking, multi-user Operating system. It is renowned for the philosophy of “Do one thing, do it well”, by having a system comprising of programs that do one task in which they are specialized and work collectively rather than a few programs that do many things inefficiently. MacOS and BSD are the most prevalent Unix descendants in the current age.

On the other hand, GNU/Linux has emerged to be the most popular and favoured Operating system that was inspired by Unix. While not a flavor of Unix, GNU/Linux has adopted all of Unix’s utilities and has largely improved upon most of its drawbacks, which gave the recursive acronym “GNU” which stands for GNU’s Not Unix.

Ubuntu is currently one of the most popular GNU/Linux distributions, which we will be using for the demonstration of the commands.

Procedure and Outputs:

1. Start a Virtual Machine of Ubuntu and launch a Terminal.

2. Print the path in which the terminal is in using **pwd (**Print Working Directory).

```
skran@skran:~/$ pwd
/home/skran/
```

3. Create a directory using mkdir (Make Directory).

```
skran@skran:~/$ mkdir test_folder
```

4. Enter the directory using cd (Change Directory).

```
skran@skran:~/$ cd test_folder
```

5. Create an empty file using touch.

```
skran@skran:~/$ touch skran.txt
```

6. List the contents of the directory you are present using ls (List).

```
skran@skran:~/$ ls
skran.txt
```

7. Print your name using echo.

```
skran@skran:~/$ echo “Richard Stallman”
Richard Stallman
```

8. Redirect the output of echo into the file you created using the redirect (**>**) operator.

```
skran@skran:~/$ echo “Richard Stallman” > skran.txt
```

9. Print the contents of that file using cat.

```
skran@skran:~/$ cat skran.txt
Richard Stallman
```

10. Delete the file using rm.

```
skran@skran:~/$ rm skran.txt
```

Conclusion:

The Unix Commands have been successfully executed on a GNU/Linux system.

Experiment 9

Aim: Demonstration of HDFS Commands

Description:

Hadoop Distributed File System: Also known as HDFS, this is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a hadoop server to hundreds of nodes.

HDFS is one of the major components of Apache Hadoop among MapReduce and YARN.

It is written in Java and provides shell commands very similar to that of Unix and GNU.

Procedure and Outputs:

1. Install hadoop and set up a single node cluster as detailed in Exps 7 and 5.
Open a terminal within your hadoop install folder.
Note: The paths mentioned below will be with respect to this location.

2. Start the hadoop cluster by running the following script.
Note: This step does not work if pdsh is installed. If there is an error, run “sudo apt remove pdsh”

```
skran@skran:~/progs/hadoop-3.3.6$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [skran]
```

3. Create a directory using -mkdir.

```
skran@skran:~/$ bin/hdfs dfs -mkdir /test
```

4. List the contents within HDFS and verify your created directory using -ls.

```
skran@skran:~/progs/hadoop-3.3.6$ bin/hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - skran supergroup          0 2023-11-06 17:36 /test
```

5. Create an empty file inside the created folder with your name using -touchz. Verify subsequently using -ls on /test.

```
skran@skran:~/$ bin/hdfs dfs -touchz /test/skran.txt
skran@skran:~/$ bin/hdfs dfs -ls /test
Found 1 items
-rw-r--r--   1 skran supergroup          0 2023-11-06 17:47 /test/skran.txt
```

6. Send a file to HDFS from your local file system using -put. Verify subsequently using -ls.

```
skran@skran:~/progs/hadoop-3.3.6$ bin/hdfs dfs -put README.txt /test
skran@skran:~/progs/hadoop-3.3.6$ bin/hdfs dfs -ls /test
Found 2 items
-rw-r--r--   1 skran supergroup        175 2023-11-06 18:05 /test/README.txt
-rw-r--r--   1 skran supergroup          0 2023-11-06 18:03 /test/skran.txt
```

7. Print the contents of a file in HDFS using -cat.

```
skran@skran:~/$ bin/hdfs dfs -cat /test/README.txt
```

```
For the latest information about Hadoop, please visit our website at:
  http://hadoop.apache.org/
and our wiki, at:
  https://cwiki.apache.org/confluence/display/HADOOP/
```

9. Copy a file from HDFS to your local system using -get.

```
skran@skran:~/$ bin/hdfs dfs -get /test/skran.txt
```

10. Delete a file on HDFS using -rm.

```
skran@skran:~/$ bin/hdfs dfs -rm /test/skran.txt
```

11. Delete a folder and its files using -rm -r (Remove Recursively)

```
skran@skran:~/$ bin/hdfs dfs -rm -r /test
```

Conclusion:

The HDFS commands have been successfully demonstrated.

It was noted that any firewalls running in the system conflicted with the nodes from communicating, thus either the required ports must be allowlisted or the firewall must be disabled.

References:

https://www.geeksforgeeks.org/hdfs-commands/

160120749051

Experiment 10

Aim: Demonstration of WordCount using Hadoop MapReduce.

Description:

Hadoop MapReduce: MapReduce is a processing module in Hadoop. It performs the desired processing on big data stored in the HDFS.

MapReduce functions by assigning fragments of data across the nodes in a Hadoop cluster. The goal is to split a dataset among the nodes, and make them work on it, achieving great parallel processing.

The parallel processing brings massive speeds and can process even Petabytes of data.

Procedure and Outputs:

1. Install hadoop and set up a single node cluster as detailed in Exps 7 and 5.
Open a terminal within your hadoop install folder.
Note: The paths mentioned below will be with respect to this location.

2. Start the hadoop cluster by running the following script.
Note: This step does not work if pdsh is installed. If there is an error, run "sudo apt remove pdsh"

```
skran@skran:~/progs/hadoop-3.3.6$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [skran]
```

3. Send the desired text file to HDFS using -put. For this example, we will send the LICENSE file present in the hadoop directory which we are present in.

```
skran@skran:~/progs/hadoop-3.3.6$ bin/hdfs dfs -put LICENSE.txt /
```

4. Run the provided WordCount program on the text file present in HDFS.

```
skran@skran:~/$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-
3.3.6.jar wordcount /LICENSE.txt /out
2023-11-06 19:07:41,975 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2023-11-06 19:07:42,034 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10
second(s).
2023-11-06 19:07:42,034 INFO impl.MetricsSystemImpl: JobTracker metrics system started
```

5. Verify the creation of output files using -ls.

```
skran@skran:~/progs/hadoop-3.3.6$ bin/hdfs dfs -ls /
Found 2 items
-rw-r--r--   1 skran supergroup      15217 2023-11-06 19:04 /LICENSE.txt
drwxr-xr-x   - skran supergroup          0 2023-11-06 19:07 /out
skran@skran:~/progs/hadoop-3.3.6$ bin/hdfs dfs -ls /out
Found 2 items
-rw-r--r--   1 skran supergroup          0 2023-11-06 19:07 /out/_SUCCESS
-rw-r--r--   1 skran supergroup       9894 2023-11-06 19:07 /out/part-r-00000
```

7. Print the contents of the output file in HDFS using -cat.

```
skran@skran:~/$ bin/hdfs dfs -cat /out/part-r-00000
"Contribution"	1
"Contributor"	1
"Derivative	1
"Legal	1
"License"	1
"Licensor"	1
"NOTICE"	1
...
```

8. Alternatively, use the -tail option to view the last few lines of the file.

```
skran@skran:~/$ bin/hdfs dfs -tail /out/part-r-00000
bility,	1
responsible	1
result	1
resulting	1
retain,	1
revisions,	1
rights	1
risks	1
royalty-free,	2
same	1
section1
...
```

Conclusion:

The WordCount program has been successfully demonstrated.

This experiment can be performed in a much more demonstrative way on an extremely large file, (GBs in size) with a cluster of 12-15 computers in a lab.

References:

https://dev.to/donaldsebleung/setting-up-a-single-node-hadoop-cluster-p0a