**- Command structure in Linux and linux commands:**

Command <options> <arguments>

echo(command) –n(option) Hello(argument)

example: print home directory echo $HOME

- Internal/built-in commands: echo, cd, pwd, mkdir etc

- External commands: mv, date, uptime, cp, runtime

- How to determine if command internal/external: type echo

- Print information on how long has the system been running for since last reboot: uptime

- Change the default reference folder (~): pushd/popd

ls options: l – long, a – all, t – time, creation order, - tr reverse order, h – human readable

whatis – 1 line description of what a command does

man <command> - documentation of the command

<command> --help

Apropos <command> - searched through manpages for keywords <command>

- Alias dt=date

- Shell used: echo $SHELL

- Change shell: sudo chsh –s(shell) /bin/[shell] [user]

- Environmental variables print: env and to set an env variable export variable=value, but you will have to declare them in ~/.profile or ~/. pam\_environment

- See the path variable: echo $PATH

- Check if the location of a command can be identified: which <command>

- Changing the bash prompt: print it first: echo $PS1, and then PS1=[……………]

- Linux Kernel version: uname –r / uname –a

- Version of the OS installed: cat /etc/\*release\* | grep ……. Release/version

- How to list and get detailed information about hardware devices attached to the linux system

dmesg – tool used to display messages from an area of the Kernel called ring buffer

udevadm info –query=path –name=/dev/sda5

udevadm monitor

- Check hardware specs: lspci/lsmem (–summary)/free –m(mb) –g(gb)/lsblk/lscpu/lshw

- Check init process: ls –l /sbin/init

- Check the runlevel(operation mode): runlevel

Runlevel 1-5 (3 non graphical, 5 graphical)

Systemd Target = runlevels

Runlevel 3 = multiuser.target = bootare de CLI

Runlevel 5 = graphical.targer = bootare grafica

- Check the operation mode (target,previous runlevel):

systemctl get-default = ls –ltr /etc/systemd/system/default.target care path de fapt este e symbolic link/shortcut catre /lib/systemd/system/graphical.target

- Change the operation mode (target, previous runlevel)

systemctl set-default [target] – example: systemctl set-default multi-user.target

The term runlevels is used in the sysV init systems. These have been replaced by systemd targets in systemd based systems.

The complete list of runlevels and the corresponding systemd targets can be seen below:

runlevel 0 -> poweroff.target

runlevel 1 -> rescue.target

runlevel 2 -> multi-user.target

runlevel 3 -> multi-user.target

runlevel 4 -> multi-user.target

runlevel 5 -> graphical.target

runlevel 6 -> reboot.target

**Package management:**

RPM – Red Hat, CentOS, Fedora – can’t solve dependencies by itself

Package install with RPM: rpm – ivh [package].rpm #install #verbose

Package uninstall with RPM: rpm –e [package].rpm

Package Upgrade with RPM: rpm –Uvh …

Package query with RPM: rpm –q [package].rpm (ex. Find exact package name for wget rpm -qa | grep wget)

Package verifying with RPM: rpm –Vf <path to file>

YUM – Yellowdog update manager – RPM based distros – Software repo (/etc/yum.repos.d) – automatic dependency resolution

yum install/remove/update/upgrade

yum repolist

yum provides [command] – verify which package should be installed for a command to work

DPKG – Debian, Ubuntu, Linux Mint, Pure OS – can’t solve dependencies by itself

Package install/upgrade with DPKG: dpkg – i [package].deb

Package uninstall with DPKG: dpkg –r [package].deb

Package list with DPKG: dpkg –l [package(#command)] fara deb

Package status with DPKG: dpkg –s [package] fara deb

Package verifying with DPKG: dpkg -p <path to file>

Apt install/remove/search/list | grep

Check file size: du (disk usage): du –sk(kilo) [file], du –sh [file] / ls –lh

- Archiving:

tar –cf [archive.tar] [file1] #c-create #f-archive name

ls –ltr [archive.tar]

tar –tf [archive.tar] list archive elements

tar –xf [archive.tar] extract contacts from tar ball

tar –zcf [archive.tar] [file1] - compress to reduce size

- Compressing/uncompressing:

bzip2 [filename.ext] / bunzip2 [filename]

gzip [filename.ext] / gunzin [filename]

xz [filename.ext] / unxz [filename]

- Read files without decompressing them with: zcat/bzcat/xzcat

- Searching for file and directories

locate [filename.ext] – may need updatedb command

find [target directory] –name [filename.ext]

grep -i [name from a file(search element)] [filename.ext] #-I nu tine cont de C/c d/D DAR/dar

grep –r “text” /home/user – search recursively for a pattern in a directory

grep –v “text” [filename] – prints lines that don’t match a pattern/string

grep –w “text” [filename] – prints the the lines that contain exactly “text”

grep –vw “text” [filename} – prints the other lines that don’t contain “text” (ex: texts)

Example file premier-league-table.txt. cat premier-league-table.txt:

1. Arsenal
2. Liverpool
3. Chelsea
4. Manchester City

grep –A1 [Arsenal] [premier-league-table.txt] (1 after argument. Argument Arsenal, Liverpool after)

1. **Arsenal**
2. Liverpool

grep –B1 [4] [premier-league-table.txt] (1 before argument. Argument 4, 3(Chelsea before))

1. Chelsea
2. **Manchester City**

grep –A[n] –B[n] – could work combined

3 Data streams created when you launch a command in linux, which through IO redirection could be redirected to text files:

Standard Input – stdin – input command

Standard Output – stdout – output of the command

Standard Error – stderr – error message of the command

Redirect STDOUT:

Echo $SHELL”text” > shell.txt – override

Echo $SHELL/”text” >> shell.txt – append (add another line)

Redirect STDERR:

cat [error message, ex: missing\_file] 2> error.txt - override

cat [error message, ex: missing\_file] 2>> error.txt – append

cat [error message, ex: missing\_file] 2> /dev/null – (bitbucket, dump standard error which we do not need to be printed on the screen)

echo $SHELL | tee shell.txt – instead of redirect > we could use | and tee to redirect and print the output – override version

echo $SHELL/”text” | tee -a shell.txt – append

Print default editor: update-alternatives --display editor

**DNS – name resolution**

Enter a name in order do resolve it:

cat >> /etc/hosts

[ip] db

Print the host name: hostname

DNS resolution made through resolv.conf which points to the DNS server:

cat /etc/resolv.conf (it shows the IP of the DNS server)

nameserver [ip, 192.168.1.100]

Default DNS order is hosts(local) and after resolv.conf which specifies the DNS Server, but the order could be changed by modifying the /etc/nsswitch.conf

Cat /etc/nsswitch.conf

Hosts: files dns #means files first (files=/etc/hosts), dns = dns server

Tools to test DNS resolution:

nslookup – query a hostname from a DNS server (does not consider the local entries from /etc/hosts)

dig – returns more details in a similar form as it is stored in the server

**NETWORKING**

See or modify interfaces for the host: **ip link** / ip link set dev eth0 up(down)

See ip addresses assigned to those interfaces: ip addr/ ip a

To see the default gateway configured: ip r

Assign IP addresses on the interfaces: ip addr add [ip/subnet] dev [dev;ex:eth0]

To permanently change the routes save them in /etc/network/interfaces

See the existing routing configuration on a system: **route (displays the kernel routing table)**

To configure a gateway that would allow communication between two different networks through a router (gateway 1 for the networking A and gateway 2 for the networking B); (example from system A to reach system B through router (gateway A)

sudo ip route add [destination network – B IP/subnet] via [gateway A IP]

sudo ip route add default/0.0.0.0/ via [gateway]

Standard networking troubleshooting:

1. Interface status (up/down) – ip link
2. Check DNS resolution – nslookup [hostname] – verify if it resolves a valid IP
3. Check connectivity – ping
4. Check route – traceroute (shows number of hops/devices between source and destination (repo server)) – it will show if there is a problem with any device in the network route between source and destination
5. Check services: netstat an | grep 80 | grep –I LISTEN – check if the HTTP process is running on port 80; netstat could be used to print information of network connections, routing tables and other network statistics

**Security and file permissions**

Account types:

> Root user (UID=0)

> System Accounts – ssh, mail (UID<100 OR between 500-1000)

> User Accounts

> Service Accounts – nginx, mercury

Information about user account is stored in /etc/passwd (unique identifier UID)

Information about group is stored in /etc/group (unique identifier GID)

Information about an user: id [user]

Information about who is logged into the system: who, last(all logged in users)

Switch user: su – or run a command for another user su –c “[command]”

/etc/sudoers – can be modified with visudo – contains the users that are allowed to use sudo for privilege escalation

Access control files: /etc/passwd, /etc/shadow (stores passwords – but are hashed) /etc/group

grep –I ^[username] /etc/passwd , unde outputul:

USERNAME:PASSWORD:UID:GID:GECOS:HOMEDIR:SHELL

grep –I ^ [username] /etc/shadow, unde outputul

USERNAME:PASSWORD:LASTCHANGE:MINAGE:MAXAGE:WARN:INACTIVE:EXPDATE

grep –I ^[username] /etc/group , unde outputul:

USERNAME:PASSWORD:GID:MEMBERS

**Managing users**

useradd [username] – must be used in root mode / userdel [username]

passwd [username] – change password – must be used in root mode

whoami – check the user id

groupadd –g [GID] [groupname] / groupdel [groupname]

example: useradd –u 1009 –g 1009 –d /home/Robert –s /bin/bash –c “Mercury Project member” bob

-u = UID

-g = GID

-d = custom home directory

-s = specify login shells

-c = custom comments

-e = expiry date

-G = create user with multiple secondary groups

**Linux File Permission:**

There are three categories:

rwx rwx r-x

owner(u) group(g) others(o)

|  |  |  |
| --- | --- | --- |
| Bit | Purpose | Octal Value |
| R | Read | 4 |
| W | Write | 2 |
| X | Execute | 1 |
| - | No permission | 0 (directory) |

0

Modifying file permissions:

**chmod <permission> file –symbolic mode**

Examples, owner(u) group(g) others(o), + add permissions, - remove permissions:

chmod u+rwx <file> -> provide full access to the owner

chmod ugo+r-x <file> provides read permissions to owner, group and others. Removes execute access

chmod o-rwx <file> - remove all access for others

chmod u+rwx, g+r-x,o-rwx <file>

**chmod <permission> file –numerical mode**

chmod 777 <<file> - read write execute for owner group others

chmod 555 <file> - read and execute permissions for owner group others

chmod 660 <file> - read and write permissions for owner and group – no permissions for others

chmod 750 <file> - read, write and execute permissions for owner, read and execute permissions for group, no permission for others

**Change the ownership and group of a file/director: chown owner:group <file>**

chown owner(if not specified, the group remains unchanged – changing only the owner) <file>

chown –R <directory> (recursive

chgrp <group> <file>

SSH commands:

ssh <hostname or IP address>

ssh <user>@<hostname or IP address>

ssh –l <user> <hostname or IP address>

- For password-less authentification:

ssh-keygen –t rsa – generates a key pair (public key /home/user/.ssh/id\_rsa.pub and private key /home/user/.ssh/id\_rsa)

ssh-copy-id <user>@<hostname or IP address> - to copy directly the public key to the server you are wanting to access

Public key is installed in the specific file /home/user/.ssh/authorized\_keys

SCP – allows you to copy files through SSH

scp <file> <remote server>:<file system where to copy, ex: /home/bob/> -> you have to have rights to write

scp –p(preserve ownership and permission of the source file)r(to copy directory)) <directory> <remote server>:<file system where to copy, ex: /home/bob/>

You can also validate this by running: netstat -ntlp | grep ssh on a server running SSH service.

**IPTABLES**

sudo iptables –L – list the default rules configured in the system – they are set to accept by default

iptables –A/-I(-A adds to the bottom, -I adds to the top of the IPTABLES <add rule – INPUT/OUTPUT> –p <protocol, tcp> –s <source ip or range(-d destination)> -dport <destination port, ex:22) –j <action to take, ACCEPT/DROP>

iptables –A INPUT –p tcp –s 172.16.238.187 -dport 22 –j ACCEPT

sudo iptables –D <OUTPUT/INPUT> < line number in iptables 5> - delete a line in IPTABLES

iptables –A INPUT –p tcp –s 172.16.238.187 -dport 22 –j ACCEPT

iptables –A INPUT –p tcp –s 172.16.238.187 -dport 80 –j ACCEPT

Exercise:

On devapp01, add an outgoing rule permitting access to port 5432 on devdb01 and HTTP access to caleston-repo-01. Once this is done, block outgoing traffic to any destination on http/https ports from devapp01

Note: caleston-repo-01 has the ip address of 172.16.238.15 and devdb01 172.16.238.11

Commands:

sudo iptables -A OUTPUT -p TCP -d 172.16.238.11 --dport 5432 -j ACCEPT

sudo iptables -A OUTPUT -p TCP -d 172.16.238.15 --dport 80 -j ACCEPT

sudo iptables -A OUTPUT -p tcp --dport 80 -j DROP

sudo iptables -A OUTPUT -p tcp --dport 443 -j DROP

Crontab –e – schedule a command at an exact hour everyday -> Cron job (crond service)

Do not use sudo with crontab command, because the job will be scheduled for the root user.

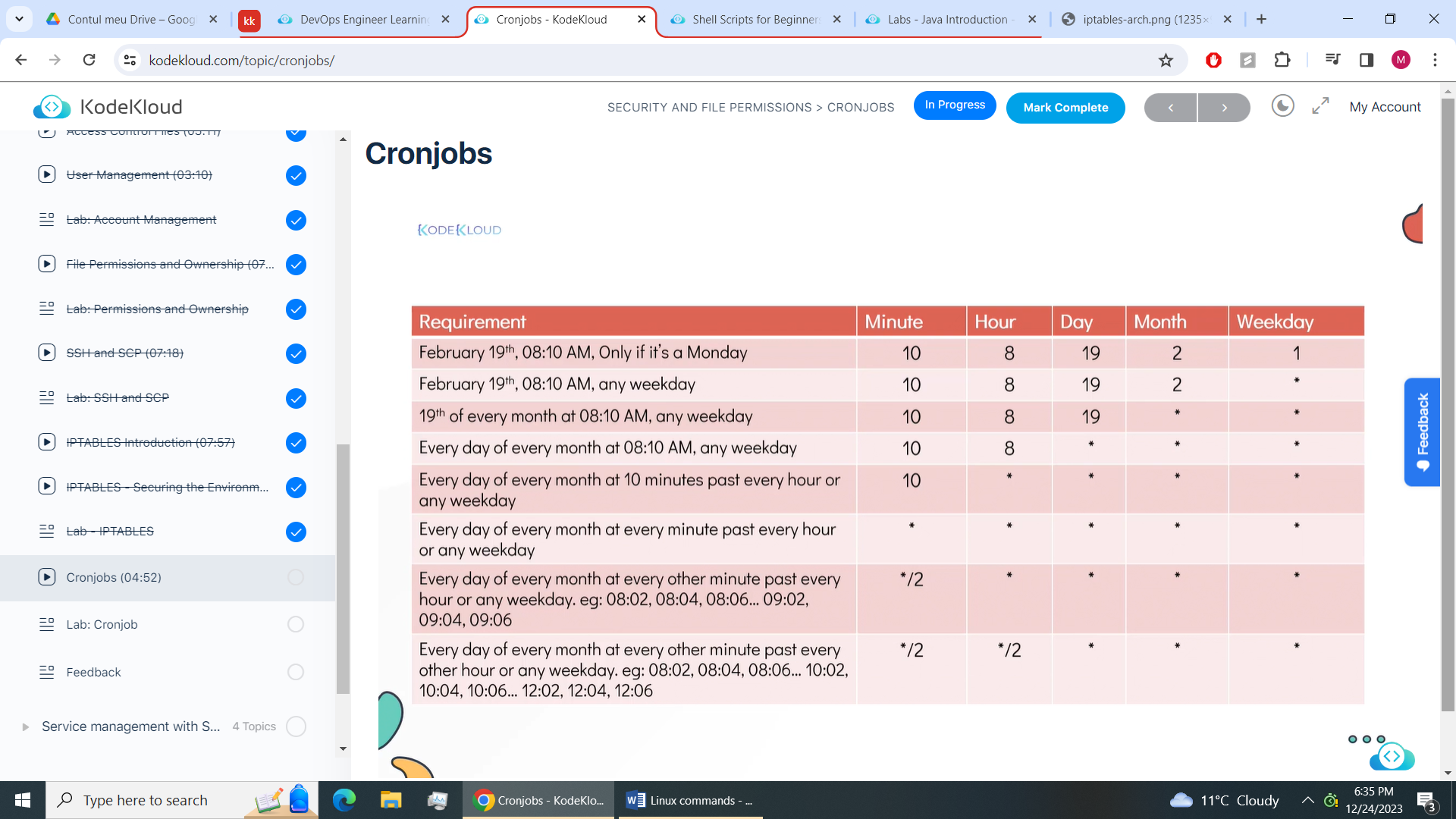
Example for 08:10 19th February Monday

Minute hour day month weekday

10 8 19 2 1

\* = any value

If minute is \* = any minute; \*/ = any 2 minutes



List all the jobs scheduled in cron: crontab –l

Check if the job was run successfully: cat /tmp/system-report.txt or tail /var/log/syslog

<https://cronjob.xyz/>

**Service Management – Systemd**

If you want to create a service you have to define it in /etc/systemd/system/[filename].service

Example: bash script /usr/bin/project-mercury.sh -> /etc/systemd/system/project-mercury.service

[Unit] #to make sure that our service is started before another service

Description=Python Django for Project Mercury #not mandatory but good practice

Docummentation=http://www………….. #not mandatory but good practice

After=postgresql.service

[Service]

ExecStart(derivative used to run a command/app)= /bin/bash /usr/bin/project-mercury.sh

User=project\_mercury #using service account

Restart=on-failure #when and how to restart the service, another option Restart=always

RestartSec=10 #10 seconds restart interval

[Install] #to allow this service to be enabled during boot

Wantedby=graphical.target

To run this service in background: sudo systemctl start project-mercury.service

To verify if the script is running in the background: systemctl status project.mercury.service

In order to modify the service you must first stop it: systemctl stop [service]

For the system to detect any changes done to the service file we have to reload the daemon:

**systemctl daemon-reload** and then we start the service **systemctl start project-mercury.service**

Systemctl is the main command used to manage services on a systemd managed server:

- manage system state

- start/stop/restart/reload

- enable/disable

- list and manage units

- list and update targets

Journalctl tool

- query systemd journal (contents of the systemd logging system called journal)

Commands:

systemctl start/stop [service]

systemctl reload [service] – used to reload the service without interrupting normal functionality

systemctl enable/disable – to enable/disable a service and make it persistent to cross reboot

systemctl status [service] -> States: active-service running/inactive-service stopped/transient states activating or deactivating/failed-crashed, error, timeout etc

systemctl daemon-reload – running this command after making changes to a service unit file reloads the system manager configuration and makes the systemd aware of the changes.

systemctl edit [service].service –-full – edit the service through this command in order to apply the changes made to the unit file immediately without running systemctl daemon-reload

**Systemctl to manage states**

systemctl get-default - see the current runlevel (target) of the system

systemctl set-default [target, ex: multi-user.target] – to change the current target to another

systemctl list-units --all – to list all the units that systemd has loaded or attempted to load (if you use it without --all flag it will print only the active units)

**journalctl** – is used when troubleshooting issues with systemd units as it checks the journal (log entries from all parts of the system) – journalctl used without flags prints all the log entries from the oldest to the newest entries

journalctl –b – prints the entries from the current boot

journalctl –u [unit, ex: [filename].service] – to see a specific unit

**Storage lessons**

See block devices (storage-SSD example) = **lsblk** or **ls –l /dev/ | grep “^b”** (shows devices who have b as a first character)

Each block device has a major and a minor number, major number which is used to identify the block device:

**1 – RAM, 3 – Hard Disk/CD ROM, 6 – Parallel printers, 8 – SCSI DISK(SD)**

Another way to list the partition table is, prints additional information: sudo fdisk –l /dev/sda

There are 3 types of disk partitions:   
- Primary partition – can be used to boot an operating system – in MBR(master boot record) there could be only 4 primary partitions – the max size per disk is 2 TB

- Extended partition – is similar to a disk drive which has a partition table which points to a logical partition – you could create an extended partition instead of the fourth primary partition

Partition scheme = partition table

GPT = GUID partition scheme – more recent than MBR, addresses the limitation in MBR and can have an unlimited number of primary disks; limited by the os itself, example: RHEL 128 maximum partitions per disk, does not have max size per disk, unless the OS installed requires MBR

gdisk /dev/[sdb] – use this command to partition disks -> press ? for help -> n to create a new partition -> w to write the partition

to verify: sudo fdisk –l /dev/[sdb] or lsblk

Disk to be usable, next steps:

- Disk partitioning – raw disk

- Create a filesystem – to write to disk/partition – defines how data is stored on a disk

- Mount filesystem to a directory – then we can read/write data to it

**Filesystems:**

Extended file system series (ext2-ext4)

Create an ext4 filesystem:

mkfs.ext4 /dev/sdb1

mkdir /mnt/ext4

mount /dev/sdb1 /mnt/ext

mount | grep /dev/sdb1 - check if the filesystem is mounted

df –hP | grep /dev/sdb1 - check if the filesystem is mounted

To make this mount to be available after booth we have to make an entry to /etc/fstab

Check the type of the filesystem(ext): sudo blkid [path, ex: /dev/vdc]

**External Storage:**

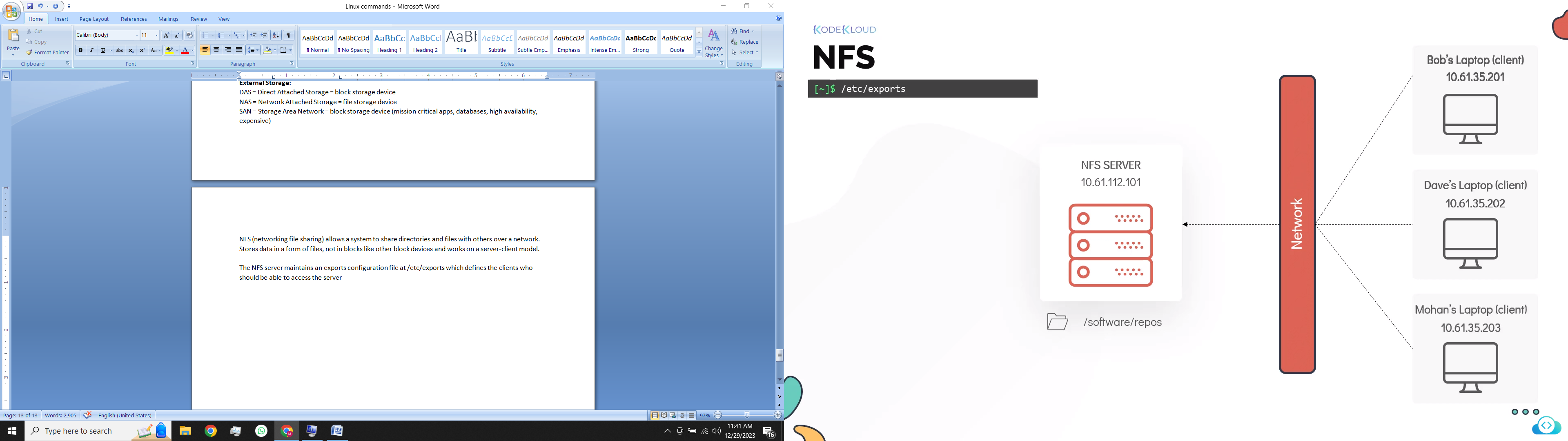
DAS = Direct Attached Storage = block storage device

NAS = Network Attached Storage = file storage device

SAN = Storage Area Network = block storage device (mission critical apps, databases, high availability, expensive)

NFS (networking file sharing) allows a system to share directories and files with others over a network. Stores data in a form of files, not in blocks like other block devices and works on a server-client model.

The NFS server maintains an exports configuration file at **/etc/exports** which defines the clients who should be able to access the server.



Export config for the above example:

/software/repos 10.61.35.201 10.61.35.202 10.61.35.203 (could be used with hostname instead of IPs, subnet, wildcard \* etc)

To export all the mounts defined in the /etc/exports file: **exportfs –a**

To manually export a directory: **exportfs –o [to which ip I would like to export]:/software/repos**

Once exported, it could be mounted on a local directory:

mount [10.61.112.101:/software/repos mnt/software/repos]

**Logical Volume Manager – LVM** – allows grouping of multiple physical volumes (hard disk or partitions-sda1,sdb1 etc) into a volume group (sda1 and sdb1). From this volume group, you can carve out logical volumes (VOL1, VOL2).

Benefits:  
- LVM allows the logical volumes to be resized dynamically as long as there is sufficient space in the volume group

Steps to install LVM:

**sudo apt-get install** **lvm2** and identify free disks or partitions and to create physical volume objects for them. A physical object volume is how LVM identifies a disk or a partition (PV)

Create a physical volume: **pvcreate /dev/sdb** ->>>>> **verify with pvdisplay**

Create a volume group(VG): **vgcreate caleston\_vg /dev/sdb** ->>>>> **verify with vgdisplay**

Create logical volumes: **lvcreate –L 1G –n vol1 caleston\_vg**

-L = linear volume – common type of logical volume – enables us to make use of multiple physical volumes if available and the volume group to create a single logical volume.

To see the logical volume: **lvdisplay / lvs**

Once the volume has been created we need to create a filesystem on it:

**mkfs.ext4 /dev/caleston\_vg/vol1**

Mount it: **mount –t ext4 /dev/caleston\_vg/vol1 /mnt/vol1**

Resize the filesystem on vol1 while it’s mounted.

1. Check for enough space in the VG: **vgs**

2. Increase the logical volume: **lvresize –L +1G –n /dev/caleston\_vg/vol1**

3. Check the size on the /mnt/vol1 **:df –hP /mnt/vol1**

4. Increase the filesystem: **resize2fs /dev/caleston\_vg/vol1**

**Troubleshooting final Lab:**

The web server is hosted on devapp01.  
  
The DB server is hosted on devdb01

Bob has done his bit when it comes to the code. Everything is set up to work properly.  
  
All you need to do is to fix some of the things he has missed while migrating the code and the database on the DEV servers.

**Task 1**:  
  
Copy the file caleston-code.tar.gz from Bob's laptop to Bob's home directory on the webserver devapp01

Solve: sudo scp caleston-code.tar.gz bob@devapp01:~/

**Task 2**:  
  
On the devapp01 webserver, unzip and extract the copied file in the directory /opt/.

Solve: sudo tar -C /opt/ -xvf caleston-code.tar.gz

**Task 3:**  
  
Delete the tar file from devapp01 webserver.

Solve: rm -rf caleston-code.tar.gz

**Task 4**:

For the client demo, Bob has installed a postgres database in devdb01.

SSH to the database devdb01 and check the status of the postgresql.service

What is the state of the DB?

Solve: SSH into devdb01 and systemctl status postgresql.service

**Task 5**:  
  
What port is postgres running on? Check using the netstat command?

Solve: sudo netstat -natulp | grep postgres | grep LISTEN

**Task 6**:  
  
It appears that Bob did not configure his app to connect a postgres database running on a different server.  
  
That explains why things are working on his laptop and not in the DEV servers.  
  
It also appears that he is using the wrong port for postgres!

1. Find the file in the directory under /opt/caleston-code that has a string matching DATABASES = {.
2. Replace the value of localhost to devdb01
3. In the same file fix the postgres port to match the port being used on devdb01

Solve: 1. grep -ir "DATABASES = {" /opt/caleston-code/ ; 2. vi and edit

**Task 7**:  
  
Now that has been set up, change the ownership of ALL files and directories under /opt/caleston-code to user mercury.

Solve: sudo chown -R mercury /opt/caleston-code/

**Task 8**:

Great! Everything should now be in order to restart the application.  
  
On the devapp01 server start the webserver again by running the command:

1. Navigate to the directory /opt/caleston-code/mercuryProject
2. Next, run the command python3 manage.py runserver 0.0.0.0:8000

Note:- Make sure to activate the virtual environment using source ../venv/bin/activate within the current project before executing python3 manage.py migrate.  
  
Something like (venv) should now be a part of the prompt.

To access the application, click on the Project Mercury tab!

Solve: 1. First activate our virtual environment with the source command

cd /opt/caleston-code

source venv/bin/activate

2. Apply database migrations:

cd /opt/caleston-code/mercuryProject

python3 manage.py migrate

Start the application

python3 manage.py runserver 0.0.0.0:8000

**Task 9**:

Well done! Now, for the final task before the client presentation.  
  
Create a new service called mercury.service with the following requirements.

1. Service name: - mercury.service, WorkingDirectory: - /opt/caleston-code/mercuryProject/, Command to run: /usr/bin/python3 manage.py runserver 0.0.0.0:8000.
2. Restart on failure and enable for multi-user.target.
3. Run as user mercury.
4. Set description: Project Mercury Web Application.

Create the unit file under /etc/systemd/system. Once done, enable and start the mercury.service.

Solve:

1. Create the service

sudo vi /etc/systemd/system/mercury.service

and insert the required data:

[Unit]

Description=Project Mercury Web Application

[Service]

ExecStart=/usr/bin/python3 manage.py runserver 0.0.0.0:8000

Restart=on-failure

WorkingDirectory=/opt/caleston-code/mercuryProject

User=mercury

[Install]

WantedBy=multi-user.target

2. Start and enable the service and check the status as follows: -

sudo systemctl enable --now mercury.service

sudo systemctl status mercury.service

SED command:

sudo sed -i 's/8080/5000/g' app.py

* **sed**: Stream editor for filtering and transforming text.
* **-i**: In-place editing. It means that the file specified (**app.py** in this case) will be edited directly.
* **'s/8080/5000/g'**: This is the substitution command in **sed**. It searches for the string "8080" and replaces it with "5000". The **g** at the end means it will perform this substitution globally, replacing all occurrences on each line, not just the first one.
* **app.py**: This is the name of the file where the substitutions will be made.

**SSL and TLS:**

**> Generate a private and public key pair: openssl genrsa –out my-bank.key 1024**

**> Request a certificate to be signed by a CA – Certificate signing request – CSR:**  
openssl req –new –key my-bank.key –out my-bank.csr

-subj “/C=US/ST=CA/O=MyOrg, Inc./CN=my-bank.com

Learning path: <https://kodekloud.com/learning-path/devops-engineer/>

Current course: <https://kodekloud.com/lessons/service-management-with-systemd/>

Second course to end: <https://kodekloud.com/topic/labs-java-introduction/7>

Next course: <https://kodekloud.com/courses/shell-scripts-for-beginners/>

Json course: https://kodekloud.com/courses/json-path-quiz/