Technical Assignment-for edrakh

this is the documentation for the assignment tasks

---

task 1:
for the first task we just create a the directory with simple mkdir command, but instead of multiple mkdir command we just create them from single command using simple RE, then touch to create the files and cat any thing to these files.
note that i use /usr/bin/env because i am running nixos on my machine now so this is the way to go as nixos have diffrennt FHS.

output:
user@user: ls project/*/
project/docs/:
README.md

project/src/:
main.py

project/test/:
test_main.py

---

task 2:
check task2.sh

output:
user@user: hostname
devops-student

---

task 3:
using ip to print and then pass it to grep to get all line sthat start with inet then filter loopback and then get the ip only by using awk.

output:
user@user: ./task3.sh
192.168.8.14/24

---

task 4:
installing mysql depend on whether by building it from the git repo or by installing it from the package manger whatever it is, for me i use nixos so by just adding these lines to the conifguration and rebuild the mysql server get installed.

```
services.mysql = {
enable = true;
package = pkgs.mariadb;
};
```

the mariadb is a fork of mysql.

the task4.sh file use mysql command with -e to run commands directly from the shell, and there is no outputs if the task run successfully.

---

task 5:
using mysqldump to backup the database.
ofcource it would be better if we encrypted the .sql file with something like gpg if the disk not encrypted, but for now this is unnecessary.

---

task 6:
for the backup.sh i used rsync which pretty good tool called rsync, rsync has a lot of options and features like compressing, but we just need the -a,–partial, and –append you can see the man page for more info, also i used -v but it isn't mandatory.

---

task 7:
although systemd timers have more features but cron is a very good tool that serve most use cases
to make a cron service in linux, first you need to install it, and then use crontab -e to edit the cron config file
in nixos we can create them easily by adding them to the configuration.nix file like that

```
services.cron =
enable = true;
systemCronJobs = [
"0 0 * * * ababa /home/ababa/edrakSoftware/task6/backup.sh /home/ababa/important/ /home/ababa/cronbackup/"
];
};
```

---

task 8:
adding the line below to Configure the SQL server to only accept connections from localhost.

bind-address=127.0.0.1

for the password checking after some research i found a good plugin for mariadb called simple_password_check check this for more info
Link
installing the plugin either by my.cnf or by running
INSTALL SONAME 'simple_password_check';

test output:
MysqlDB:: CREATE USER 'test_plugin'@'localhost' IDENTIFIED BY 'bad';
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements (simple_password_check)

---

task 9:
there are a lot of monitoring tools like a tool called btop written in cpp and it has good ui and configured at the run time, for more advanced usage there is (nmon-written in c-, and glances) both have the ability to export the logs to external files for later analizing.
for the gpu there is nice tool called nvtop

for the sql logs adding this lines to the my.cnf enable the logging
general_log = 1
general_log_file = /home/ababa/.mysql/logs

also by default you can see the logs by systemctl tools
user@user: journalctl -u mysql.service

---

last task:
the security in linux servers is heavely depend on the use-case, but here is general best practices that suited for almost every use-case.

- remove/disable/block everything that you don't use like open ports,apps and services.
- make sure there is a good password policy by checking with tools like cracklib witch clracklib-check tool
  or by enforcing a good pasword by any policykit like libpam-cracklib or by using pam_pwquality library.
  Arch wiki link

- it's not a bad idea to disable ssh password login and enable key login in theory if you have a good passwords this wouldn't make any diff but it's a good additionlayer of protection also you if you can create ssh ip whitelist that block any ip not in the list from using ssh
  also changing ssh port will help with bot or blind bruteforce attacks, but wouldn't help that much with an organized attack.
- control and limit who can use sudo or su, disabling su completly is good idea to consider.
- using firewall like ufw
- using AppArmor which is a Linux kernel security module that allows the system administrator to restrict programs capabilities like network access,file permission and other capabilities.
- using or compiling a kernel with Security-Enhanced Linux or aka SElinux.
- also using a vulnerability scanner like Nessus or lynix.
- using an Intrusion prevention software like Fail2ban witch will help with bruteforce attacks.
- setting hidepid to 2 for the /proc folder can be a good option for some use-cases but in general the default behavier is more suitable for most use-cases.
- for the physical security see Arch wiki