**Server Side:**

How to configure MySQL server to start automatically on boot?

sudo update-rc.d mysql defaults

How to check and display the status of the MySQL service?

sudo systemctl status mysql

**Task 1: Advanced User and Group Configuration**

**1.1 Nested Group Creation:**

How to create a group?

sudo groupadd group\_name

How to assign a user to a group?

sudo usermod -aG group\_name username

//-aG: append the user to a group without removing them from other groups

How to remove a user from a group?s

sudo gpasswd -d username groupname

What is sudoers group?

-We assign a user to sudoers group to allow them to execute commands with root privileges using sudo command

How to assign a user to a sudoers group?

sudo usermod -aG sudo username

How to list the groups:

cat /etc/group

How to check user’s groups:

groups username

How to delete a group?

sudo groupdel group\_name

**1.2 Shared Directories with ACLs:**

What is the difference between chmod and ACL (Access control list)?

-chmod is only able to give permission to user owner, group owner, and others, while ACL is able to give permission to special users or special groups.

To set ACL for a file or directory:

setfacl -m u:name:permission file\_name (we can change the ‘u’ to ‘g’ to set for groups)

-m: to modify or add ACL

u: user

name: user or group name

permission: r for read, w for write, x for execute

To view the ACL for a file or directory:  
getfacl file\_name

**Task 2: Security Hardening:**

**2.1 SSH Key-Based Authentication:**

How to disable password authentication?

1. Open the SSH configuration file: sudo nano /etc/ssh/sshd\_config
2. Disable password authentication for the user by uncommenting PasswordAuthentication and PubkeyAuthentication and set their values to PasswordAuthentication no, PubkeyAuthentication yes
3. Save changes and reload the SSH using: sudo systemctl restart sshd

Why do we disable password authentication?

**1. Prevent Brute Force Attacks**

* Password-based authentication is vulnerable to brute force attacks, where an attacker repeatedly tries different password combinations.

**2. Enforce Secure Authentication Methods**

* Forcing the user to use SSH keys, which are cryptographically stronger than passwords.

**3. Restrict Access for Specific Users**

* If some users require SSH access while others don’t, you can control who can log in and how they authenticate.

**4. Prevent Unauthorized Access**

* If an attacker steals a user’s password, they still can’t access the server if password authentication is disabled.

How ssh works?

When we generate ssh key, 2 types of keys are generated, Public key and Private key, the Private key cant be shared, it stays inside the device itself, while the Public key can be shared to everyone ( will be shared to the server). So lets say I want to access the Server\_VM using Client\_VM, I have to log in Client\_VM and generate an ssh key (Public key and Private key) then share Client\_VM public key to the server, so whenever Client\_VM want to access the Server\_VM, the Server\_VM will send a random message to Client\_VM, then the Client\_VM encrypt this message using his Private key and send it to the Server\_VM, the Server\_VM will verify the authenticator of the message using the stored public key, if valid then the authentication is successful and now Client\_VM can access the Server\_VM.

How to generate and deploy SSH keys for dev\_lead1 to access VM1?

Log in using Client\_VM and generate ssh key using: ssh-keygen -t rsa -b 4096

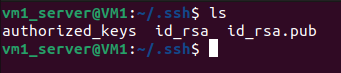
-t rsa: specify the RSA Algorithm

-b 4096: Sets the key size to 4096 bits for stronger security.

After generating the key, it will ask where to save the key 🡪 Don’t change anything, save it in the default file.

Then it will ask to set passphrase (Optional) to protect it using password 🡪 you can skip it

Once you are done, the private and public keys will be generated and saved inside /home/user/.ssh



Authorized\_keys: stores the **public keys** of users who are allowed to log in via SSH key authentication.

id\_rsa: stores the private key of the user

id\_rsa.pub: stores the public key of the user which will be shared

Now we have to share the public key of the Client\_VM to the Server\_VM, we have many ways to share it but the best one using the ssh-copy-id command. How?

In the Client\_VM, write: ssh-copy-id dev\_lead1@Server\_IP\_ADDRESS

Server\_IP\_ADDRESS: the remote host that you would like to connect to

Dev\_lead1: the account where your public key will be shared to

* To know the Server IP you can use the command: hostname -I or ip a

To connect to the user using SSH:

Inside Client\_VM write: ssh dev\_lead1@Server\_IP\_ADDRESS

* For the first time, your local device doesn’t recognize the remote host, so it will prompt whether you want to continue or not: type yes and enter

**For more details: take a look at** [**How to Create an SSH Key in Linux: Easy Step-by-Step Guide | DigitalOcean**](https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server#step-2-copying-an-ssh-public-key-to-your-server)

* 1. **Automatic Security Updates:**

How to enable automatic security updates using unattended-upgrade?

* Log in to the server and install the package using : sudo apt-get install unattended-upgrades -y
* Also install the update dash notifier common package for automatic reboots using sudo apt install update-notifier-common -y
* Edit the 50 unattended dash upgrade file using:

sudo nano /etc/apt/apt.conf.d/50unattended-upgrades

* Remove the // and change the false to true from the line that contains: Unattended-Upgrade::Automatic-Reboot “false”;
* Add the following line to log updates:

Unattended-Upgrade::LogFile "/var/log/security\_updates.log";

Issue the command Cat /var/run/reboot-required to see if a reboot is required, if yes, reboot the machine and that’s it

* 1. **Configure Message of the Day (MOTD):**
* Open the motd file using the command: Sudo nano /etc/motd
* Add the custom message “Welcome to the Ubuntu administration Lab”

**Task 3: System Monitoring & Automation:**

**3.1** **Metric Collection Script:**

**1. Variables and Setup**

LOG\_DIR="/var/operations/monitoring"

TIMESTAMP=$(date +"%Y%m%d\_%H%M%S")

LOG\_FILE="$LOG\_DIR/metrics\_$TIMESTAMP.log"

sudo mkdir -p "$LOG\_DIR"

* **LOG\_DIR**: Directory where logs are stored.
* **TIMESTAMP**: Generates a unique timestamp (format: YYYYMMDD\_HHMMSS).
* **LOG\_FILE**: Full path for the output log file.
* **sudo mkdir -p**: Ensures the log directory exists or creates it.

**2. Collecting System Metrics**

The collect\_metrics function gathers:

1. **CPU and Memory Usage**

top -b -n1 | head -10

* Captures a snapshot of the top processes consuming resources.

1. **Disk I/O Statistics**

sudo iostat -x 1 1

* Displays extended disk I/O statistics.

1. **Top 5 Resource-Heavy Processes**

ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu | head -n 6

* Lists processes sorted by CPU usage (top 5 plus the header line).

1. **Service Status (MySQL and SSH)**

sudo systemctl is-active mysql

sudo systemctl is-active ssh

* Checks if these services are running and logs their statuses.

**3. Restarting Services if Down**

The restart\_service function handles service restarts if inactive:

1. **Service Check and Restart**

if ! sudo systemctl is-active --quiet "$service"; then

sudo systemctl restart "$service"

* If the service is inactive, the script restarts it.

1. **Verification**

if sudo systemctl is-active --quiet "$service"; then

echo "INFO: $service restarted successfully."

else

echo "ERROR: Failed to restart $service."

* Confirms if the restart was successful or logs an error.

1. **Handling Apache Variants**

if sudo systemctl list-units --type=service | grep -q "apache2.service"; then

restart\_service apache2

elif sudo systemctl list-units --type=service | grep -q "httpd.service"; then

restart\_service httpd

**3.2 File Activity Monitor:  
1. Variables and Setup**

WATCH\_DIR="/projects/development"

LOG\_FILE="/var/log/file\_changes.log"

sudo mkdir -p "/var/log"

* **WATCH\_DIR**: Directory to monitor.
* **LOG\_FILE**: Output log file for file activity.
* **mkdir -p**: Ensures the log directory exists.

**2. Monitoring File Activity**

inotifywait -m -r -e create,modify,delete --format '%T %w%f %e %u' --timefmt '%Y-%m-%d %H:%M:%S' "$WATCH\_DIR" | while read event

* **-m**: Keeps the process running continuously.
* **-r**: Monitors directories recursively.
* **-e**: Filters events (create, modify, delete).

**Task 4: MySQL User Management and Database Exploration:**

After installing the mysql-server, you can open it using:  
mysql -u root -p

It will ask for password, by default it will not let you in unless you rewrite the code using sudo:

Sudo mysql -u root -p

To change the password of the user:

ALTER USER 'root'@'localhost' IDENTIFIED BY 'NewPassword';

Then exit

IDENTIFIED BY: ASSIGN PASSWORD TO THE USER

To show all the databases stored:

show databases;

If you want to use a certain database:

Use DATABASE\_NAME;

To show all the tables inside a database:

show tables;

To create a new database:

Create database DATABASE\_NAME

To create a table inside a database:

Similar commands to SQL-Developer (we can use similar syntax for queries, updating, deleting ….)

**4.1 MySQL User Creation:**

**4.2 User Authentication and Verification:**

**4.3 Database and Table Exploration:**

**4.4 Logging and Audit Trail:**

**Task 5: Network Connectivity Monitoring:**

Creates a directory PingDirectory in the user’s home directory if it doesn’t exist.

Captures the current timestamp in the format YYYYMMDD\_HHMMSS.

Pings two client VMs (192.168.18.16 and 192.168.18.17) using ping -c 10 -s 500 -i 10 $ClientVM\_IP

with specific parameters:

* Sends 10 packets (-c 10).
* Uses a packet size of 500 bytes (-s 500).
* Waits 10 seconds between packets (-i 10).

Logs the ping results to ~/PingDirectory/TimeStamp.txt, including a timestamped header, separators, and the ping output for both VMs.

Sets the file permissions to 600 (owner-only read/write).

**Client Side:**

**Task 1: Development Team (VM2):**

**1.1 Login Attempt Monitoring:**

**1.2 Permission Cleanup Script:**

We want to change the permission of every file in VM2 that has permission 777 (all permissions) to (read, write and execute for owner ONLY)

How to do that:

First, we need to set a variable called “LOGFILE” to the log file we want to create our logs in:

LOGFILE="perm\_changes.log"

To set the search path for the script to be the default home directory of the user that is running the script:

SEARCH\_PATH="$HOME"

To create the log file (if it does not exist)

touch "$LOGFILE"

to check if current user has permission to write to the log file

if [ ! -w "$LOGFILE" ]; then

echo "Error: Cannot write to log file $LOGFILE"

exit 1

fi

Now, to find all the files in the home directory that has permission 777 and set it to 777 we write:

find "$SEARCH\_PATH" -type f -perm 777 | while read -r file; do

chmod 700 "$file"

echo "$(date '+%Y-%m-%d %H:%M:%S') - Changed permissions of $file from 777 to 700" >> "$LOGFILE"

done

-type f is used to ensure that the type is a file, while read -r file is used to iterate over all the files, echo "$(date '+%Y-%m-%d %H:%M:%S') is used to add a timestamp on the log file of every modification and >> "$LOGFILE" is added to the store the logs in the log file

To print a statement that shows on the terminal that the script was executed successfully:

echo "Permission changes complete. Check $LOGFILE for details."

**Task 2: Operations Team (VM3):**

**2.1 Resource Reporting Script:**

In this task, we want to collect system resource metrics on VM3 and securely transfer the report to VM1 every hour.

How to do that:

**1. First, we need to define some variables:**

* TIMESTAMP=$(date +"%Y%m%d\_%H%M%S")

Stores the current date and time in YYYYMMDD\_HHMMSS format.

* REPORT\_FILE="/tmp/resource\_report\_${TIMESTAMP}.txt"

Defines the file path for storing the resource report.

* VM1\_IP="192.168.x.x"

Specifies VM1’s IP address.

* VM1\_USER="dev\_lead1"

The username that will transfer the report to VM1.

* VM1\_DEST\_PATH="/var/operations/reports"

The directory on VM1 where reports are stored.

* DEST\_PATH="${VM1\_USER}@${VM1\_IP}:${VM1\_DEST\_PATH}/"

Combines user, IP, and destination path for secure copying.

**2. After that we need to create the report file with restricted permissions to securely store system metrics:**

* touch "$REPORT\_FILE" 2>/dev/null || { echo "Error: Cannot create report file"; exit 1; }

Creates the report file in /tmp/.

* chmod 640 "$REPORT\_FILE"

Sets permissions (read/write for owner, read-only for group, no access for others).

**3. Then we start writing report content; adds a timestamped header to organize and track reports efficiently:**

* echo "===== Resource Report for VM3 (Operations Team) - $TIMESTAMP =====" > "$REPORT\_FILE"

**4. The next step is the process tree: which we will capture a hierarchical view of running processes to monitor system activity:**

* pstree -p >> "$REPORT\_FILE" 2>/dev/null

this will captures the process hierarchy with process IDs.

* If it fails, logs an error.

**5. After that we need to identify and log orphaned processes that may cause system resource issues (Zombie Processes):**

* ps aux | awk '$8=="Z" {print "PID: " $2 " - " $11}' >> "$REPORT\_FILE" 2>/dev/null

to find and log zombie processes.

* ZOMBIE\_COUNT=$(ps aux | awk '$8=="Z"' | wc -l)

counts the total zombie processes.

* If no zombies exist, logs "No zombie processes found."

**6. Then record current CPU load and memory consumption to detect performance bottlenecks:**

* top -bn1 | head -n 3 >> "$REPORT\_FILE" 2>/dev/null

captures CPU usage snapshots.

* free -h >> "$REPORT\_FILE" 2>/dev/null

logs memory usage.

**7. Lists the most CPU- and memory-intensive processes for troubleshooting (Top 5 Resource-Consuming Processes):**

* ps -eo pid,ppid,user,%cpu,%mem,cmd --sort=-%cpu | head -n 6 >> "$REPORT\_FILE" 2>/dev/null

Logs the top 5 processes by CPU usage.

* ps -eo pid,ppid,user,%cpu,%mem,cmd --sort=-%mem | head -n 6 >> "$REPORT\_FILE" 2>/dev/null

Logs the top 5 processes by memory usage.

**8. Securely copy the report to VM1 for centralized storage and analysis.:**

* scp -o "StrictHostKeyChecking=no" "$REPORT\_FILE" "$DEST\_PATH" 2>/dev/null

Securely copies the report to VM1.

* If successful, logs "Successfully copied report to VM1".
* If it fails, logs an error and exits.

**9. Finally, delete the local report file to prevent unnecessary storage accumulation.**

* rm "$REPORT\_FILE"

**2.2 Quota Enforcement:**

In this task, we want to monitor disk usage in the /shared directory on VM3 and enforce user quotas for dev\_lead1 and ops\_lead1. If users exceed their limits, the script will send an email alert to the administrator.

**How to do that:**

1. **First, we need to define some variables:**
   * SHARED\_DIR="/shared"  
     Specifies the directory that will be monitored for disk usage.
   * ADMIN\_EMAIL="admin@qu.edu.qa"  
     The email address that will receive alerts when users exceed their quota limits.
   * LOG\_FILE="/var/log/quota\_check.log"  
     Defines the log file path where all quota-related activities will be recorded.
   * TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')  
     Captures the current timestamp to include in logs and alerts.
2. **After that, we set the quota limits (soft and hard) for users to enforce disk usage restrictions:**
   * DEV\_LEAD1\_HARD=$((5 \* 1024 \* 1024))  
     Sets a **hard limit** of **5GB** for dev\_lead1 (enforcement threshold).
   * DEV\_LEAD1\_SOFT=$((6 \* 1024 \* 1024))  
     Sets a **soft limit** of **6GB** for dev\_lead1 (warning threshold).
   * OPS\_LEAD1\_HARD=$((3 \* 1024 \* 1024))  
     Sets a **hard limit** of **3GB** for ops\_lead1.
   * OPS\_LEAD1\_SOFT=$((4 \* 1024 \* 1024))  
     Sets a **soft limit** of **4GB** for ops\_lead1.
3. **To ensure the log file and the monitored directory exist to prevent errors:**
   * touch "$LOG\_FILE" 2>/dev/null || { echo "Error: Cannot create log file"; exit 1; }  
     Ensures that the log file exists, and if not, creates it.
   * [ -w "$LOG\_FILE" ] || { echo "Error: Cannot write to log file"; exit 1; }  
     Ensures the script has the required permissions to write to the log file.
   * if [ ! -d "$SHARED\_DIR" ]; then mkdir -p "$SHARED\_DIR"; chmod 755 "$SHARED\_DIR"; fi  
     Checks if the /shared directory exists; if not, creates it with appropriate permissions.
4. **We defined a function to send email alerts when a user exceeds their quota:**
   * send\_email() is a function that takes two arguments: the email subject and message content. The function sends an email using the mail -s command, logging any failures, If the email fails, it logs an error message in the quota log file.
5. **We also define a function to check the disk usage of a user and compare it with quota limits:**
   * The function calculates the disk usage of the user in KB, It then converts this usage to **GB** for readability, then logs the user’s disk usage, along with their quota limits, If a user exceeds the **soft limit**, a **warning** is logged, and an email alert is sent, If a user exceeds the **hard limit**, a **critical alert** is logged, and an email notification is sent to the administrator.
6. **Then we execute the quota check for each user:**
   * We need to call the function check\_quota for both dev\_lead1 and ops\_lead1, to ensure that their disk usage is checked against their assigned limits.
7. **Finally, exit the script after execution to indicate a successful run:**
   * exit 0