2025

Hardened MySQL Deployment: Encrypted Communication, Access Control, and Backup Strategy

MYSQL HARDENING VIA TLS, ACCESS RESTRICTION, AND ENCRYPTED BACKUPS

TEBOGO MATSEDING

1st Login

Installing MySQL Secure Installation

Create a Certificate Authority (CA)

Create the MySQL Server Certificate (Step 1)

Create the MySQL Server Certificate (Step 2)

Create the MySQL Server Certificate (Step 3)

Create the MySQL Server Certificate (Step 4)

Verify SSL/TLS certs

Configure MySQL to Use SSL

Restart MySQL

Verify SSL is Active

Test Remote Admin can connect via SSL Encryption

Installing Fail2Ban

Edit Local Jail Config File

Verify Jail is Active

Installed Auditd

Auditd Status

Track writes/changes to MySQL directory

Log every sudo command run

Log Review Using ausearch

Creating New Server for Backups

Installing MySQL-client on Backup Server

Added Backup Server to the UFW

Create a MySQL user for Backup Server

Granting Backup Server Privileges

Backup Server Connecting to MySQL

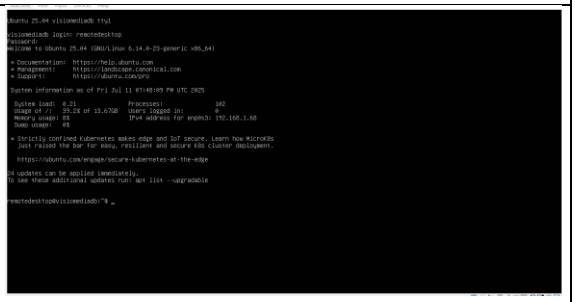
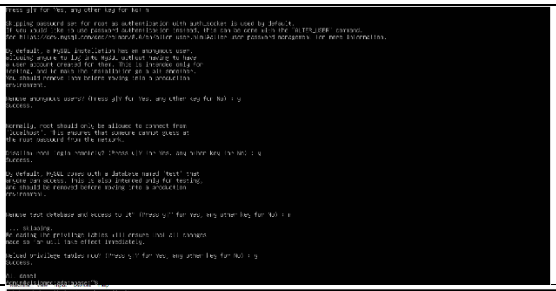
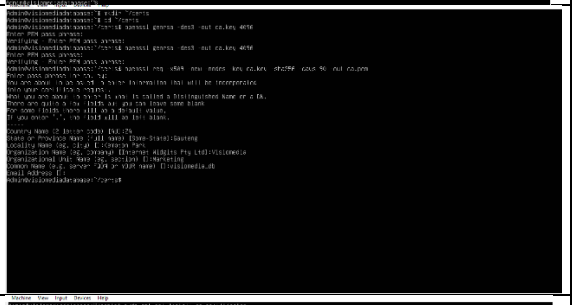
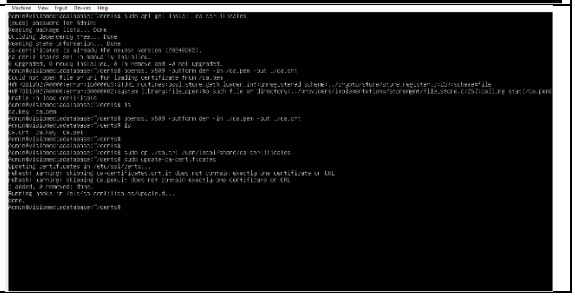
Create mysqldump script

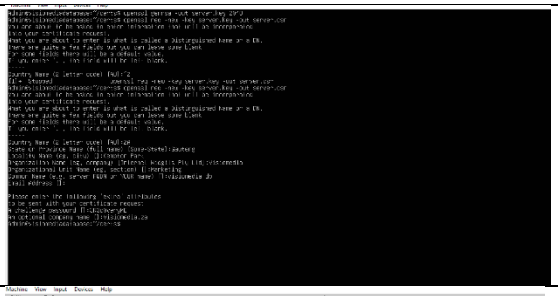
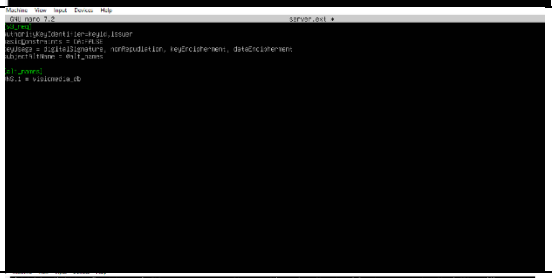
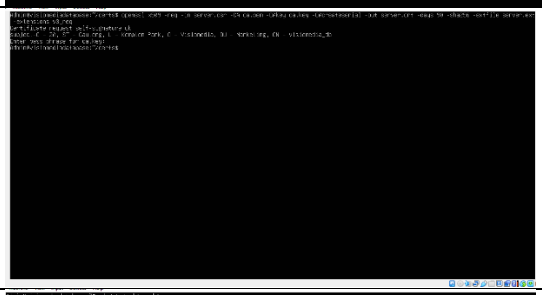
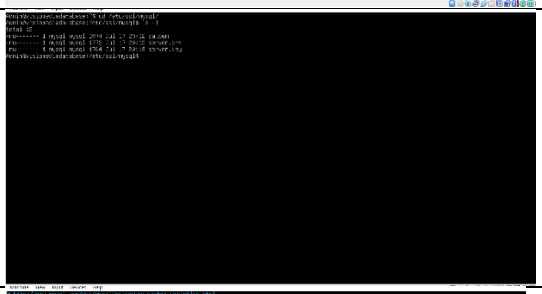
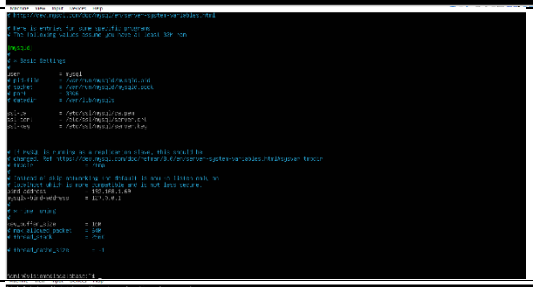
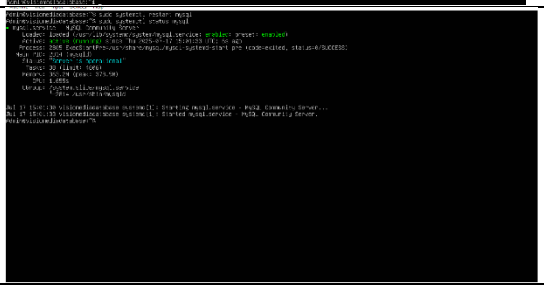
Test Script

Set crontab -e to automate backup

Crontab -l to verify it has been set

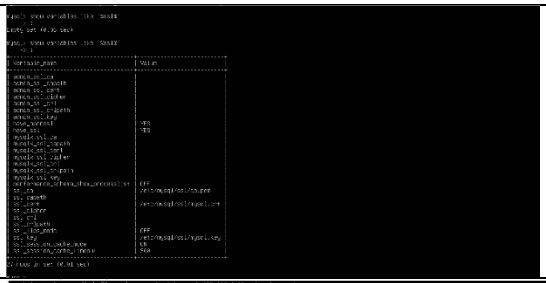
START

***NOTE** This is a continuation from my previous project "**Secure Headless Database Infrastructure Using VirtualBox"** if you haven't seen it, please click the link on the right of the table and it will direct you straight there, however if you have already seen it lets continue on	
1st Login Started by logging into the Remote Admin Server via SSH. This server acts as the secure entry point into the internal network and is the only machine allowed to communicate directly with the MySQL database server. From here, all configuration and monitoring tasks were performed.	
Installing MySQL Secure Installation After installing MySQL, I ran the secure installation wizard. It's a good way to clean up the default settings I removed anonymous users, disabled root access over the network. Making sure MySQL didn't come with any open doors.	
Create a Certificate Authority (CA) Before enabling SSL, I needed a Certificate Authority. This is what signs all the other certificates in the setup it is kind of like creating my own internal trust system.	
Create the MySQL Server Certificate Generated a private key for the MySQL server. This key stays on the server and identifies it securely during SSL handshakes.	

Create the MySQL Server Certificate step 2 Created a certificate signing request (CSR) based on the private key. This file is basically a formal request to the CA to get a proper certificate for the server.	
Create the MySQL Server Certificate step 3 Signed the CSR using my CA to produce the actual server certificate. This is what MySQL uses to prove its identity to clients.	
Create the MySQL Server Certificate step 4 Locked down the file permissions on all certificate files, then placed them in a secure location where MySQL could access them without leaving them exposed.	
Verify SSL/TLS certs Double-checked that the certificate chain was valid and all the files were working together properly, just to verifying everything was trusted and not expired or mismatched.	
Configure MySQL to Use SSL Added the certificate paths to MySQL's config file so it knows where to look when SSL connections are requested. This basically tells MySQL to enforce secure communication.	
Restart MySQL Restarted the MySQL service to load all the new SSL settings.	

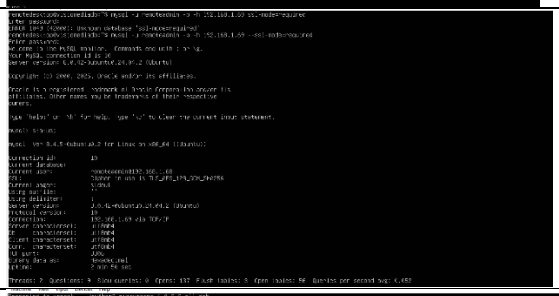
Verify SSL is Active

Connected to the MySQL server and confirmed that SSL was active by checking the variables. Everything showed up as expected, so encryption was working.



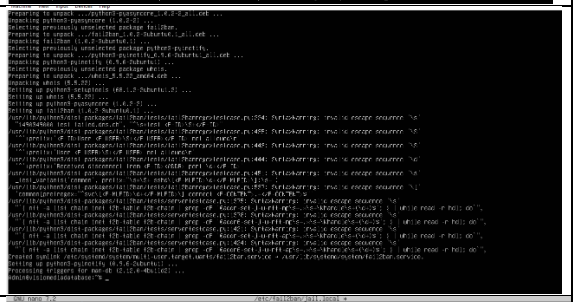
Test Remote Admin can connect via SSL Encryption

Ran a test from a remote machine using the right SSL certificates. The goal here was to make sure that no one can connect without valid encryption the connection worked only when all certs were correct.



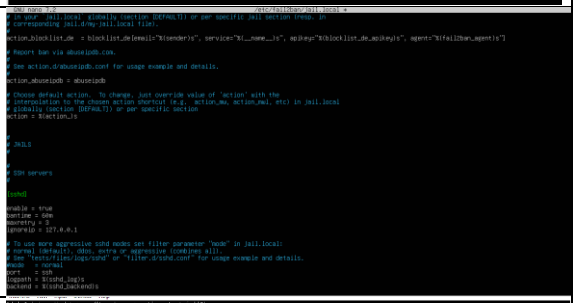
Installing Fail2Ban

Installed Fail2Ban to protect against brute-force attacks. It reads log files and blocks IPs that fail too many times. Simple but super effective.



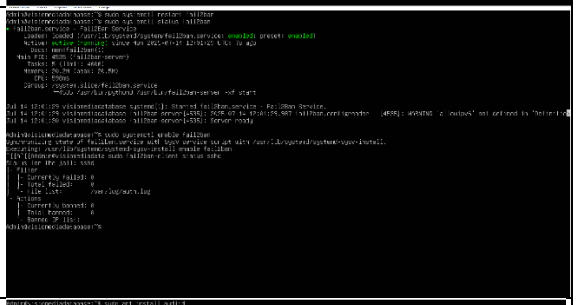
Edit Local Jail Config File

Tweaked the jail config to monitor MySQL login attempts specifically. This included adjusting the ban time and the number of allowed failures.



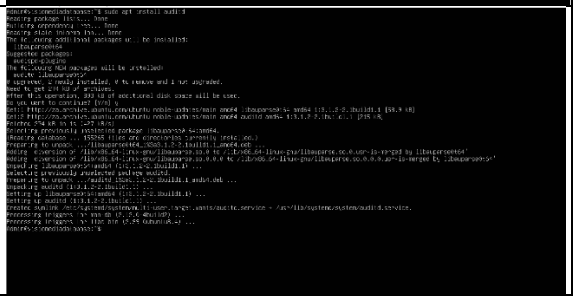
Verify Jail is Active

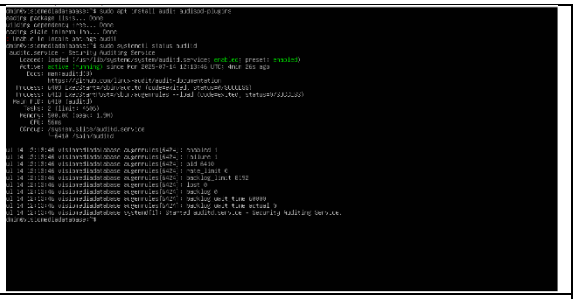
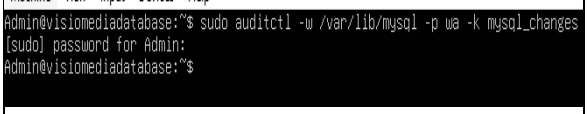
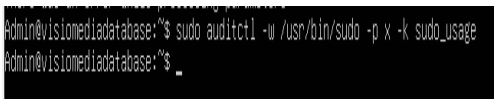
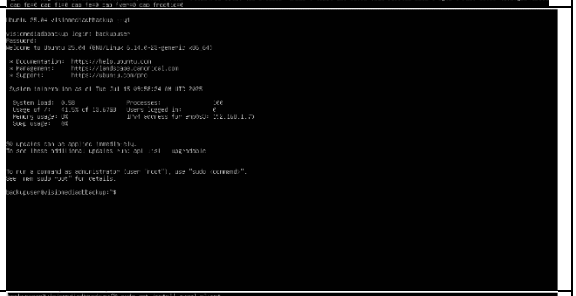
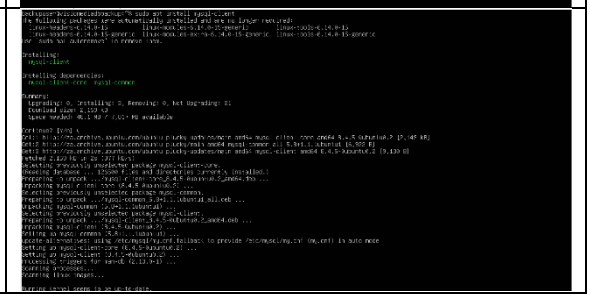
Confirmed that the jail was enabled and running. The MySQL filter was active and ready to take action if someone tried to mess around.

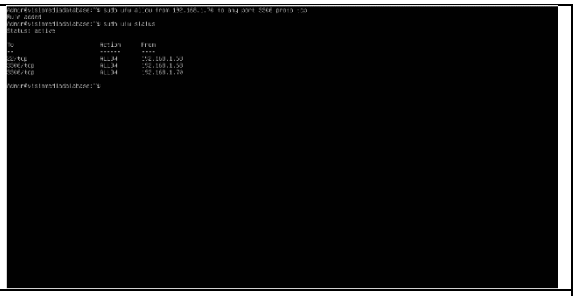
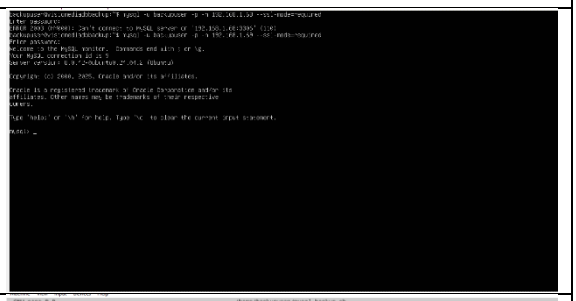
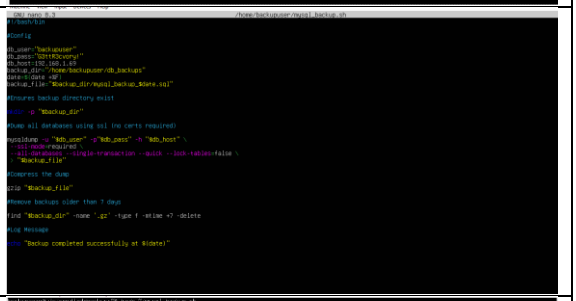
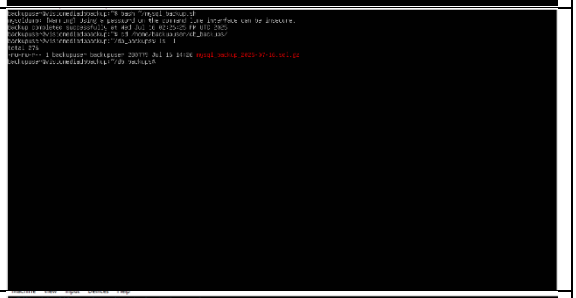
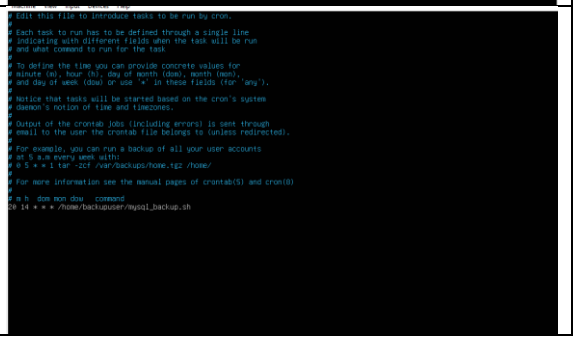


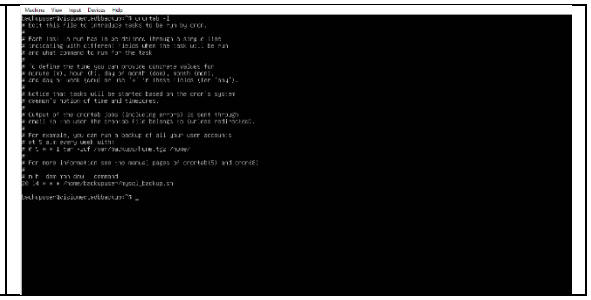
Installed Auditd

Installed Auditd so I could track important system events, especially related to MySQL file access and sudo commands.



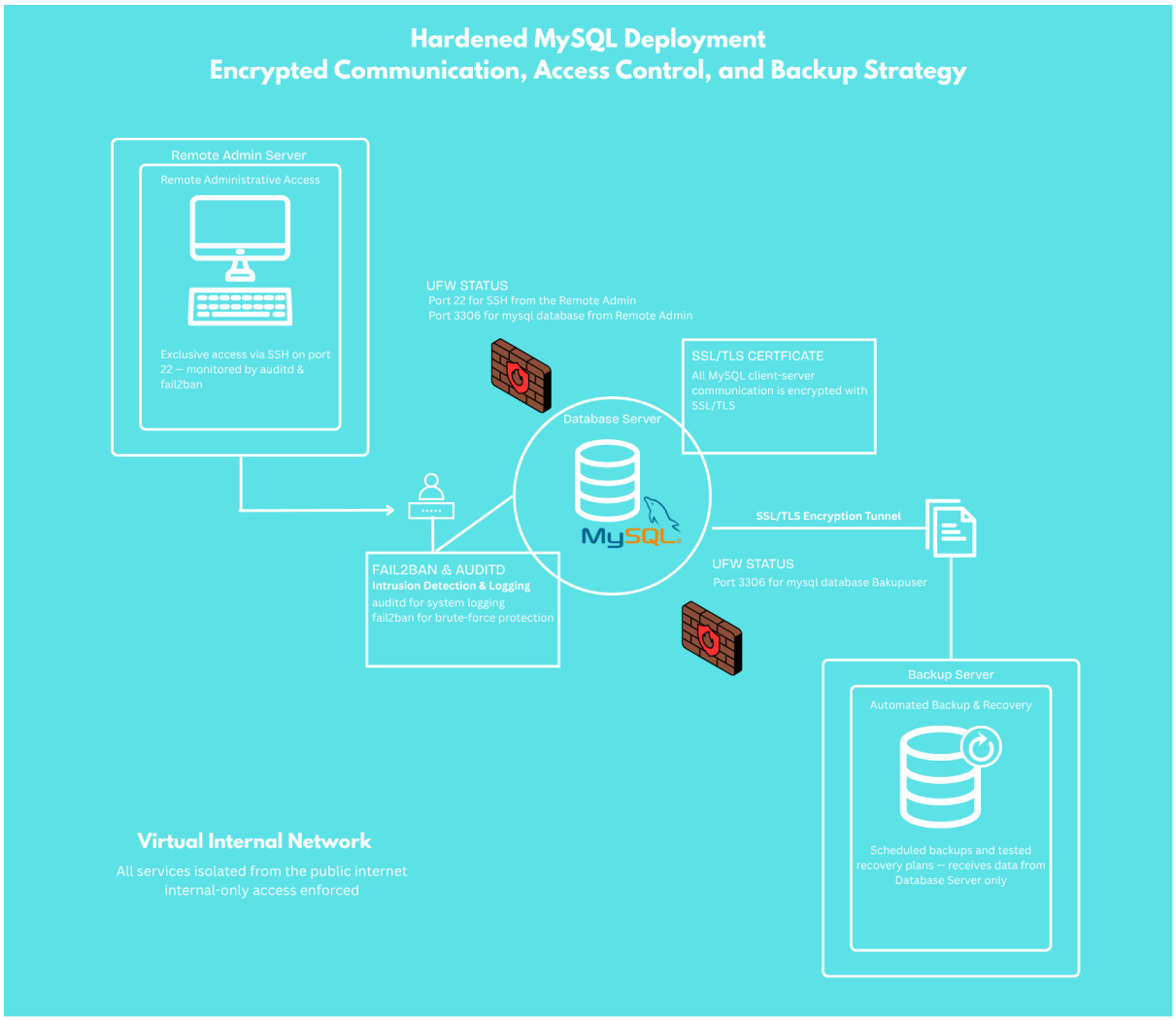
Auditd Status Made sure the Auditd service was active and running without errors. It had to be fully operational to start logging events.	
Track writes/changes to MySQL directory Set up a rule in Auditd to monitor any write or change operations inside the MySQL data directory. If anything changes in that folder, it gets logged — no exceptions.	
Log every sudo command run Added another Auditd rule to log all commands run with sudo. This helps track exactly who is doing what with elevated privileges.	
Log Review Using ausearch Used ausearch to pull up logs based on the rules I set earlier. This let me see who accessed MySQL directories and which sudo commands were executed. Super useful for auditing	
Creating New Server for Backups Spun up a second server strictly for backups. It runs headless as well, and is fully isolated except for the IP whitelist we allow through the firewall.	
Installing MySQL-client on Backup Server Only installed the MySQL client on the backup server. No database hosting, just the ability to connect to the main server and pull backups.	

Added Backup Server to the UFW On the MySQL server, I updated the firewall to only allow MySQL connections from the backup server's IP address.	
Create a MySQL user for Backup Server Created a new MySQL user specifically for backups. This user only works from the backup server's IP and is forced to use SSL to connect.	
Granting Backup Server Privileges Gave the backup user just enough privileges to read and export data — nothing more. It can't modify anything, just dump the database.	
Backup Server Connecting to MySQL Tested the connection using the client-side certificates. Confirmed that the backup server could only connect if SSL was used, which was exactly what I wanted.	
Create mysqldump script Wrote a bash script on the backup server that uses mysqldump to export the full database and save it with a timestamp. This makes it easy to keep track of backup versions.	
Test Script Ran the script manually to make sure it works. The output was clean, the file was generated, and everything looked good.	
Set Crontab -e to automate backup Added the script to the system's crontab so it runs automatically at 2:30 PM every day. Set it and forget it.	

Crontab -l to verify it has been set Listed the current cron jobs to confirm the backup task was properly registered.	

Project Overview

This deployment was all about locking down a MySQL environment inside a private virtual network. I used a Remote Admin Server to handle all database management, with tight firewall rules allowing only necessary ports. MySQL connections were secured using SSL/TLS encryption, and system activity was monitored with Fail2Ban and Auditd to catch intrusions and log sensitive actions. A second server handled automated, encrypted backups, pulling data securely through an SSL tunnel. Everything stays off the public internet — internal-only communication is enforced end to end.



Conclusion

The MySQL infrastructure I built is already secure and isolated, but there's always room to push things further. Here are five key improvements I'd consider adding next, all focused specifically on tightening security:

Enable Multi-Factor Authentication (MFA) for SSH Access

Adding 2FA to the Remote Admin Server's SSH login would make brute-force or credential-based attacks much harder to pull off.

Implement Centralized Log Aggregation & Alerting

By collecting logs from all servers into a single dashboard, it becomes easier to detect threats in real time and respond faster.

Off-Site Encrypted Backup Storage

Right now, backups are stored locally. Pushing encrypted copies to a remote location ensures recovery is possible even if the local server is compromised or damaged.

Encrypt MySQL Data at Rest

Encrypting the storage volumes or data directories protects sensitive information even if someone gets direct disk access.

Set Up Certificate Expiry Monitoring

Keeping an eye on SSL certificate expiry dates helps avoid downtime or accidental unencrypted connections when certs silently expire.