VOID

SYSTEM ADMINISTRATION

Infrastructure Deployment & Troubleshooting

Sprint 1 Technical Report

Yahya El Omari

Encadré par: Mr. Hamza Bahlaouane

February 8, 2026

CONTENTS

INTRODUCTION

This report documents the complete lifecycle of deploying a Laravel LEMP stack on Debian 11 (Bullseye). Unlike typical deployment guides, this report chronicles the *actual* deployment experience, including the challenges encountered, troubleshooting processes, and architectural decisions made in real-time.

The deployment followed a non-linear path with several pivots:

- Infrastructure Provisioning: Vagrant-based Debian 11 VM with port forwarding
- Network Security Implementation: IPtables + Fail2Ban (with backend compatibility issues)
- Web Server Deployment: Nginx with 3rd party module investigation
- Database Evolution: SQLite → MySQL 8.0 → Redis integration

This report prioritizes technical accuracy over idealized narratives, documenting both successful implementations and the debugging processes that led to them.

VIRTUAL ENVIRONMENT & SSH ACCESS

2.1 PROVISIONING: VAGRANT INITIALIZATION

The deployment environment was provisioned using Vagrant with VirtualBox as the provider. The configuration emphasized both development convenience (port forwarding, synced folders) and production-like networking.

2.1.1 VAGRANTFILE CONFIGURATION

Listing 2.1: Vagrantfile

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/bullseye64"
  config.vm.network "private_network", ip: "192.168.56.10"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "forwarded_port", guest: 3306, host: 33060

  # Synced folder for Laravel
  config.vm.synced_folder "./", "/var/www/myApp",
    owner: "www-data",
    group: "www-data"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
    vb.cpus = 2
  end

  # Provisioning script
  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y git zip unzip
  SHELL
end
```

Network Strategy:

- Port 80 → 8080: Web server access via `http://localhost:8080`
- Port 3306 → 33060: MySQL access from host for database management tools

2.1.2 VM INITIALIZATION

Listing 2.2: VM Provisioning

```
# Initialize and start the VM
vagrant up

# Verify VM status
vagrant status
# Should show: running (virtualbox)
```

2.2 ACCESS CONTROL: SSH CONFIGURATION

Secure Shell access was configured using modern cryptographic standards and identity-based authentication.

2.2.1 SSH KEY GENERATION

Listing 2.3: ED25519 Key Pair Generation

```
# Generate ED25519 key pair with specific identity
ssh-keygen -t ed25519 -C "y.elomari@void.fr" -f ~/.ssh/id_ed25519

# Verify key generation
ls -la ~/.ssh/id_ed25519*
```

Algorithm Selection: ED25519 was chosen over RSA for shorter key length, faster operations, and modern cryptographic best practices.

2.2.2 SSH CLIENT CONFIGURATION

Listing 2.4: SSH Config File (/.ssh/config)

```
Host labubu
    HostName 172.16.160.130
    User labubu
    IdentityFile ~/.ssh/id_ed25519
    ServerAliveInterval 60
    Port 22
    IdentitiesOnly yes
```

Configuration Directives:

- HostName: Server IP address (172.16.160.130)
- User: Dedicated non-root user account (labubu)
- ServerAliveInterval 60: Maintains persistent development sessions
- IdentitiesOnly yes: Prevents SSH from trying other keys

2.2.3 PUBLIC KEY DEPLOYMENT

Listing 2.5: Deploying Public Key

```
# Copy public key to VM using ssh-copy-id
ssh-copy-id -i ~/.ssh/id_ed25519.pub labubu@172.16.160.130

# Test connection
ssh labubu
```

2.2.4 CONNECTION VERIFICATION

Listing 2.6: SSH Connection Test

```
# Test connection
ssh labubu

# Verify identity
whoami          # Output: labubu
hostname        # Output: bullseye
ip addr show    # Verify 172.16.160.130 is present

# Test sudo access
sudo whoami     # Output: root
```

NETWORK SECURITY (IPTABLES & FAIL2BAN)

3.1 FIREWALL STRATEGY

A strict "deny by default" firewall policy was implemented using IPtables to minimize the attack surface.

3.1.1 IPTABLES CONFIGURATION

Listing 3.1: IPtables Ruleset

```
#!/bin/bash
# Flush existing rules
sudo iptables -F

# Set default policies to DROP
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT

# Allow loopback traffic
sudo iptables -A INPUT -i lo -j ACCEPT

# Allow established connections
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# Allow SSH, HTTP, HTTPS
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Verify rules
sudo iptables -L -v -n
```

3.1.2 PERSISTENCE CONFIGURATION

Listing 3.2: Making Rules Persistent

```
# Install iptables-persistent
```

```
sudo apt-get install -y iptables-persistent

# Save rules
sudo netfilter-persistent save
```

3.2 INTRUSION PREVENTION: FAIL2BAN

Fail2Ban provides dynamic protection against brute-force attacks by monitoring logs and automatically blocking suspicious IP addresses.

3.2.1 INSTALLATION

Listing 3.3: Fail2Ban Installation

```
sudo apt-get install -y fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

3.2.2 THE FAIL2BAN CHAIN MYSTERY

Observed Issue: After installation, the expected `f2b-sshd` chain did not appear in IPtables output.

Root Cause: Debian 11 introduced `nftables` as the default firewall backend, but Fail2Ban defaults to the `iptables` backend. The compatibility layer doesn't always synchronize chains between nftables and legacy iptables views.

Listing 3.4: Diagnosing the Backend Issue

```
# Check which iptables binary is in use
ls -la /usr/sbin/iptables
# Output: /usr/sbin/iptables -> xtables-nft-multi

# Check actual nftables rules
sudo nft list ruleset
```

3.2.3 RESOLUTION

Listing 3.5: Switch to Legacy IPtables

```
# Install legacy iptables
sudo apt-get install -y iptables

# Update alternatives
sudo update-alternatives --set iptables /usr/sbin/iptables-legacy
sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy

# Restart Fail2Ban
sudo systemctl restart fail2ban

# Verify chain appears
sudo iptables -L -v -n | grep f2b-sshd
```

3.2.4 FAIL2BAN CONFIGURATION

Listing 3.6: /etc/fail2ban/jail.local

```
[DEFAULT]
bantime = 3600
findtime = 600
maxretry = 3

[sshd]
enabled = true
port = 22
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
```

WEB LAYER & 403 FORBIDDEN RESOLUTION

4.1 NGINX DEPLOYMENT

4.1.1 INSTALLATION

Listing 4.1: Nginx and PHP 8.2 Installation

```
# Install Nginx
sudo apt-get install -y nginx

# Add PHP 8.2 repository (Sury)
sudo apt-get install -y lsb-release ca-certificates apt-transport-https
sudo wget -O /etc/apt/trusted.gpg.d/php.gpg \
    https://packages.sury.org/php/apt.gpg
echo "deb https://packages.sury.org/php/ $(lsb_release -sc) main" | \
    sudo tee /etc/apt/sources.list.d/php.list

# Install PHP 8.2 with extensions
sudo apt-get update
sudo apt-get install -y php8.2-fpm php8.2-cli php8.2-mysql \
    php8.2-sqlite3 php8.2-xml php8.2-curl php8.2-mbstring \
    php8.2-zip php8.2-redis
```

4.2 DEBUGGING LOG: THE 403 FORBIDDEN ERROR

4.2.1 SYMPTOM DOCUMENTATION

Listing 4.2: 403 Forbidden Error

```
curl -I http://192.168.56.10

# Output:
HTTP/1.1 403 Forbidden
Server: nginx/1.18.0
```

4.2.2 INVESTIGATION: PERMISSIONS

Listing 4.3: Permission Verification

```
# Check ownership
ls -la /var/www/myApp/public/
# Output: drwxrwxr-x www-data www-data

# Test read access
sudo -u www-data cat /var/www/myApp/public/index.php
# SUCCESS: File content displayed
```

Conclusion: Permissions were correct.

4.2.3 INVESTIGATION: ERROR LOGS

Listing 4.4: Error Log Analysis

```
sudo tail -f /var/log/nginx/error.log

# Output:
# directory index of "/var/www/myApp/public/" is forbidden
```

Key Insight: The error indicates no index file was found matching the `index` directive.

4.2.4 ROOT CAUSE: MISSING INDEX.PHP

3rd party modules were not the cause. The issue was identified as a missing `index.php` entry in the Nginx index directive.

Laravel's entry point is `index.php`, not `index.html`. When Nginx received a request for `/`, it couldn't find an appropriate index file and returned 403 Forbidden.

4.3 FINAL CONFIGURATION

Listing 4.5: /etc/nginx/sites-available/default

```
server {
    listen 80 default_server;
    root /var/www/myApp/public;
```

```
    # The critical fix for Laravel routing
    index index.html index.php index.htm index.nginx-debian.html;

    server_name _;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.2-fpm.sock;
    }
}
```

4.3.1 DEPLOYMENT

Listing 4.6: Applying Configuration

```
# Test syntax
sudo nginx -t

# Reload Nginx
sudo systemctl reload nginx

# Verify
curl -I http://192.168.56.10
# Expected: HTTP/1.1 200 OK
```

DATABASE & PERFORMANCE INTEGRATION

5.1 LARAVEL INITIATION

5.1.1 INITIAL SETUP

Listing 5.1: Laravel Environment Setup

```
cd /var/www/myApp
composer install --optimize-autoloader
cp .env.example .env
php artisan key:generate
```

5.2 PHASE 1: SQLITE IMPLEMENTATION

5.2.1 INITIAL ATTEMPT

Listing 5.2: SQLite Configuration

```
# .env configuration
DB_CONNECTION=sqlite

# Create database file
touch /var/www/myApp/database/database.sqlite
```

5.2.2 MIGRATION FAILURE

Listing 5.3: SQLite Error

```
php artisan migrate
```

```
# ERROR: could not find driver (SQL: PRAGMA foreign_keys = ON;)
```

Root Cause: Missing `php8.2-sqlite3` extension.

5.2.3 RESOLUTION

Listing 5.4: Installing SQLite Support

```
sudo apt-get install -y php8.2-sqlite3
sudo systemctl restart php8.2-fpm

# Verify
php -m | grep sqlite
# Output: pdo_sqlite, sqlite3
```

5.3 PHASE 2: MYSQL 8.0 IMPLEMENTATION

5.3.1 REPOSITORY CHALLENGE

Listing 5.5: GPG NO_PUBKEY Error

```
sudo apt-get update

# ERROR:
# GPG error: NO_PUBKEY 467B942D3A79BD29
# The repository is not signed
```

5.3.2 MODERN GPG KEY SOLUTION

Listing 5.6: Repository-Specific Keyring

```
# Download and install GPG key
wget -O - https://repo.mysql.com/RPM-GPG-KEY-mysql-2022 | \
    gpg --dearmor | \
    sudo tee /usr/share/keyrings/mysql.gpg > /dev/null

# Set permissions
sudo chmod 644 /usr/share/keyrings/mysql.gpg
```

Listing 5.7: /etc/apt/sources.list.d/mysql.list

```
deb [signed-by=/usr/share/keyrings/mysql.gpg] \
    http://repo.mysql.com/apt/debian/ bullseye mysql-8.0
```

Security Benefit: The MySQL GPG key can only verify packages from the MySQL repository, limiting potential security impact if the key is compromised.

5.3.3 MYSQL INSTALLATION

Listing 5.8: MySQL 8.0 Installation

```
sudo apt-get update
sudo apt-get install -y mysql-server

# Secure installation
sudo mysql_secure_installation
```

5.3.4 DATABASE SETUP

Listing 5.9: Database and User Creation

```
-- Create database
CREATE DATABASE myapp_db
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;

-- Create dedicated user
CREATE USER 'laravel_user'@'localhost'
  IDENTIFIED WITH mysql_native_password BY 'SecurePassword';

-- Grant privileges
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, INDEX,
      DROP, ALTER, LOCK TABLES, EXECUTE
ON myapp_db.* TO 'laravel_user'@'localhost';

FLUSH PRIVILEGES;
```

5.3.5 LARAVEL MYSQL CONFIGURATION

Listing 5.10: .env Configuration

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
```

```
DB_PORT=3306
DB_DATABASE=myapp_db
DB_USERNAME=laravel_user
DB_PASSWORD=SecurePassword
```

5.3.6 MIGRATION SUCCESS

Listing 5.11: Running Migrations

```
php artisan migrate

# Output:
# Migration table created successfully.
# Migrating: 2014_10_12_000000_create_users_table
# Migrated:  2014_10_12_000000_create_users_table (67.23ms)
...
```

5.4 PHASE 3: REDIS INTEGRATION

5.4.1 INSTALLATION

Listing 5.12: Redis Installation

```
sudo apt-get install -y redis-server php8.2-redis
sudo systemctl restart php8.2-fpm
```

5.4.2 SECURITY CONFIGURATION

Listing 5.13: /etc/redis/redis.conf

```
bind 127.0.0.1 ::1
protected-mode yes
requirepass YourStrongPassword
maxmemory 256mb
maxmemory-policy allkeys-lru
```

5.4.3 LARAVEL CONFIGURATION

Listing 5.14: .env Redis Configuration

```
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=YourStrongPassword
REDIS_PORT=6379

CACHE_DRIVER=redis
SESSION_DRIVER=redis
QUEUE_CONNECTION=redis
```

5.4.4 VERIFICATION

Listing 5.15: Testing Redis Integration

```
php artisan tinker

# Test cache
Cache::put('test_key', 'test_value', 600);
Cache::get('test_key');
# Output: "test_value"
```

SSL/TLS ENCRYPTION & HTTPS MIGRATION

6.1 LOCALHOST ENCRYPTION STRATEGY

6.1.1 CERTIFICATE GENERATION REQUIREMENTS

Challenge: Let's Encrypt was not viable for this deployment due to the absence of a publicly accessible FQDN (Fully Qualified Domain Name).

Technical Constraint: ACME (Automatic Certificate Management Environment) protocol challenges require:

- HTTP-01 Challenge: Public HTTP endpoint accessible at port 80
- DNS-01 Challenge: Control over DNS records for domain validation
- TLS-ALPN-01 Challenge: Public TLS endpoint on port 443

Since this deployment operates on a local/private network (172.16.160.130), none of these challenge types can be satisfied.

Solution: Generate a self-signed RSA 2048-bit certificate for localhost development and testing.

6.1.2 SELF-SIGNED CERTIFICATE GENERATION

Listing 6.1: RSA 2048-bit Self-Signed Certificate

```
# Generate private key and self-signed certificate
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout /etc/ssl/private/myApp-selfsigned.key \
    -out /etc/ssl/certs/myApp-selfsigned.crt \
    -subj "/C=FR/ST=IDF/L=Paris/O=VOID/OU=DevOps/CN=localhost/emailAddress=
        y.elomari@void.fr"

# Set restrictive permissions
sudo chmod 600 /etc/ssl/private/myApp-selfsigned.key
sudo chmod 644 /etc/ssl/certs/myApp-selfsigned.crt

# Verify certificate details
openssl x509 -in /etc/ssl/certs/myApp-selfsigned.crt -text -noout
```

Certificate Parameters:

- Algorithm: RSA 2048-bit (industry standard for self-signed certificates)
- Common Name (CN): localhost (ensures handshake consistency)
- Email: y.elomari@void.fr (administrative contact)
- Validity: 365 days
- Key Files:
  - Private Key: `/etc/ssl/private/myApp-selfsigned.key`
  - Certificate: `/etc/ssl/certs/myApp-selfsigned.crt`

Security Notes:

- `-nodes`: No DES encryption on private key (for automated server restart)
- `-x509`: Directly generate certificate instead of CSR
- `chmod 600`: Restricts private key access to root only

6.2 NGINX SECURITY HARDENING

6.2.1 DUAL-SERVER BLOCK ARCHITECTURE

The Nginx configuration was restructured to implement defense-in-depth with automatic HTTPS enforcement.

HTTP SERVER BLOCK (PORT 80)

Listing 6.2: HTTP Block - 301 Permanent Redirect

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name localhost 172.16.160.130;

    # Global HTTP to HTTPS redirect
    return 301 https://$server_name$request_uri;
}
```

Traffic Logic:

- All HTTP requests receive `301 Moved Permanently` status

- Browsers cache the redirect, reducing future HTTP attempts
- `$request_uri` preserves query strings and path

HTTPS SERVER BLOCK (PORT 443)

Listing 6.3: /etc/nginx/sites-available/default - SSL Configuration

```
server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;

    server_name localhost 172.16.160.130;
    root /var/www/myApp/public;

    # SSL Certificate Configuration
    ssl_certificate /etc/ssl/certs/myApp-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/myApp-selfsigned.key;

    # Modern SSL/TLS Configuration
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:
        ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384';
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    # Security Headers
    add_header Strict-Transport-Security "max-age=31536000;
        includeSubDomains" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;

    # Laravel routing
    index index.html index.php index.htm index.nginx-debian.html;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    # PHP-FPM Socket Bridge
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.2-fpm.sock;

        # Additional security
        fastcgi_param HTTPS on;
        fastcgi_param HTTP_SCHEME https;
    }
}
```

Security Enhancements:

- TLS 1.2/1.3 Only: Disabled legacy TLS 1.0/1.1

- Strong Cipher Suites: ECDHE with AES-256-GCM for forward secrecy
- HSTS: Strict-Transport-Security header forces HTTPS for 1 year
- FastCGI HTTPS Parameters: Ensures Laravel detects secure connection

6.2.2 DEPLOYMENT & VERIFICATION

Listing 6.4: Enabling SSL Configuration

```
# Test Nginx syntax
sudo nginx -t

# Reload configuration
sudo systemctl reload nginx

# Verify HTTP redirect
curl -I http://localhost
# Expected: HTTP/1.1 301 Moved Permanently
# Expected: Location: https://localhost/

# Verify HTTPS endpoint (allow self-signed)
curl -Ik https://localhost
# Expected: HTTP/2 200 OK
# Expected: strict-transport-security header present
```

6.3 AUTOMATED CERTIFICATE MAINTENANCE

6.3.1 PRODUCTION-READY RENEWAL STRATEGY

While self-signed certificates don't require renewal infrastructure in the same way as Let's Encrypt, establishing a renewal workflow simulates production readiness.

MONTHLY RENEWAL SCRIPT

Listing 6.5: /etc/cron.monthly/ssl-renewal

```
#!/bin/bash
# SSL Certificate Renewal Script
# Simulates production certbot renewal workflow

CERT_FILE="/etc/ssl/certs/myApp-selfsigned.crt"
LOG_FILE="/var/log/ssl-renewal.log"
DAYS_THRESHOLD=30

# Check certificate expiration
```

```
EXPIRY_DATE=$(openssl x509 -in "$CERT_FILE" -noout -enddate | cut -d= -f2)
EXPIRY_EPOCH=$(date -d "$EXPIRY_DATE" +%s)
CURRENT_EPOCH=$(date +%s)
DAYS_REMAINING=$(( ($EXPIRY_EPOCH - $CURRENT_EPOCH) / 86400 ))

echo "[$(date)] Certificate expires in $DAYS_REMAINING days" >> "$LOG_FILE"

# Renew if within 30-day window
if [ $DAYS_REMAINING -lt $DAYS_THRESHOLD ]; then
    echo "[$(date)] Renewing certificate..." >> "$LOG_FILE"

    # Regenerate certificate
    openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
        -keyout /etc/ssl/private/myApp-selfsigned.key \
        -out /etc/ssl/certs/myApp-selfsigned.crt \
        -subj "/C=FR/ST=IDF/L=Paris/O=VOID/OU=DevOps/CN=localhost/
            emailAddress=y.elomari@void.fr" \
        >> "$LOG_FILE" 2>&1

    # Reload Nginx
    systemctl reload nginx

    echo "[$(date)] Renewal complete" >> "$LOG_FILE"
else
    echo "[$(date)] No renewal needed" >> "$LOG_FILE"
fi
```

SCRIPT DEPLOYMENT

Listing 6.6: Installing Renewal Script

```
# Create script
sudo nano /etc/cron.monthly/ssl-renewal

# Make executable
sudo chmod +x /etc/cron.monthly/ssl-renewal

# Test execution
sudo /etc/cron.monthly/ssl-renewal

# Verify log
tail -f /var/log/ssl-renewal.log
```

Production Equivalence:

This script structure mirrors the `certbot renew` workflow:

- 30-Day Window: Matches Let's Encrypt renewal window
- Automated Execution: Monthly cron ensures regular checks
- Logging: Centralized audit trail in `/var/log/`
- Zero-Downtime: Nginx reload preserves active connections

Migration Path: When transitioning to production with a public FQDN:

Listing 6.7: Future Let's Encrypt Integration

```
# Replace self-signed workflow with certbot
sudo apt-get install -y certbot python3-certbot-nginx

# Automated certificate acquisition and Nginx configuration
sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com

# Certbot creates its own renewal cron in /etc/cron.d/certbot
```

6.4 SSL/TLS VERIFICATION

6.4.1 FINAL SECURITY VALIDATION

Listing 6.8: Comprehensive SSL Health Check

```
# 1. Verify certificate validity
openssl x509 -in /etc/ssl/certs/myApp-selfsigned.crt -noout -dates
# Check: notAfter date is 365 days from generation

# 2. Test TLS handshake
openssl s_client -connect localhost:443 -servername localhost < /dev/null
# Check: Protocol version (TLSv1.2 or TLSv1.3)
# Check: Cipher suite matches configured preferences

# 3. Verify HTTP to HTTPS redirect
curl -I http://localhost
# Expected: 301 Moved Permanently

# 4. Validate HTTPS endpoint
curl -Ik https://localhost
# Expected: 200 OK
# Expected: Strict-Transport-Security header

# 5. Check Laravel HTTPS detection
php artisan tinker
>>> request()->secure()
# Expected: true
```

6.4.2 SECURITY POSTURE

Achieved Security Controls:

- Encryption in Transit: All traffic encrypted via TLS 1.2/1.3

- Forward Secrecy: ECDHE key exchange prevents retroactive decryption
- HSTS Enforcement: Browsers automatically upgrade to HTTPS
- Security Headers: Protection against XSS, clickjacking, MIME sniffing
- Automated Renewal: Simulated production certificate lifecycle

System Status: Fully operational and secured with enterprise-grade SSL/TLS encryption.

CONCLUSION

7.1 DEPLOYMENT SUMMARY

This Sprint 1 deployment successfully established a production-ready LEMP stack environment running Laravel on Debian 11.

Infrastructure Layer:

- Vagrant-based Debian 11 VM
- ED25519 SSH key authentication
- ServerAliveInterval for persistent sessions

Security Layer:

- IPtables firewall with DROP default policy
- Fail2Ban intrusion prevention (legacy iptables backend)
- Repository-specific GPG key isolation
- SSL/TLS encryption with RSA 2048-bit self-signed certificate
- HTTP to HTTPS automatic redirection (301 Permanent)
- HSTS enforcement with modern cipher suites

Application Stack:

- Nginx web server with correct index.php directive
- PHP 8.2-FPM with encrypted socket bridge
- MySQL 8.0 with dedicated application user
- Redis for caching and session management

7.2 KEY TROUBLESHOOTING INSIGHTS

7.2.1 FAIL2BAN CHAIN VISIBILITY

Issue: Chains not visible due to nftables/iptables backend conflict.

Solution: Switched to legacy iptables using `update-alternatives`.

7.2.2 NGINX 403 FORBIDDEN

Issue: 403 error despite correct permissions.

Root Cause: Missing `index.php` in index directive.

Lesson: Error messages can be misleading; "forbidden" meant "no index file found."

7.2.3 MYSQL GPG KEY ERROR

Issue: NO_PUBKEY error when adding MySQL repository.

Solution: Repository-specific keyring with `[signed-by=...]` directive.

Lesson: Modern package management prioritizes security through isolated trust relationships.

7.2.4 DATABASE EVOLUTION

Phase 1: SQLite failed due to missing extension.

Phase 2: Migrated to MySQL 8.0 for production features.

Phase 3: Added Redis for performance optimization.

Lesson: Incremental deployment allows targeted troubleshooting.

7.3 SYSTEM STATUS

Listing 7.1: Final Health Check

```
# All services running
sudo systemctl status nginx php8.2-fpm mysql redis-server fail2ban

# Application accessible
curl -I http://192.168.56.10
# Expected: HTTP/1.1 200 OK

# Database connectivity
php artisan migrate:status

# Redis operational
redis-cli ping
```

```
# Expected: PONG
```

Final Status: The Laravel LEMP stack is fully operational, secured with enterprise-grade SSL/TLS encryption, and ready for application development and deployment.