WordPress Operations Runbook

For [CUSTOMIZE: Domain/Instance/Environment]

By [CUSTOMIZE]

Version 3.1 — April 22, 2026

Contents

Table of Contents

Emergency Quick-Reference Card

Use this table to quickly find solutions to common issues.

Symptom	Go To	First Command
Site returns 500 error	Section 10.2	`tail /var/log/php-errors.log` (if PHP/WP-CLI fails: check error log first)
Site is slow/not responding	Section 10.5	`top` and `free -h`
Missing database tables	Section 11.3	`wp db check`
Plugin causing crashes	Section 10.2	`wp plugin deactivate --all`
Need to restore from backup	Section 11.2	Check backup integrity first
WordPress cannot write to disk	Appendix A	`ls -ld /home/wordpress/public_html/wp-content/uploads`
SSL certificate expired	Section 3.5	`openssl x509 -in [cert] -noout -text`
Users locked out	Section 5.5	`wp user list --role=administrator --format=table`
Hacked/malware suspected	Section 10.3	Isolate site immediately, see escalation
Database won't start	Section 11.2	`systemctl status mysql`

Section 1: Document Information

1.1 Purpose

This runbook provides comprehensive operational guidance for managing, maintaining, securing, and troubleshooting a WordPress installation in production. It covers:

- Daily maintenance and monitoring
- Emergency response procedures

- Backup and disaster recovery
- Security hardening and incident response
- Deployment workflows and rollback procedures

1.2 Audience

This document is intended for:

- WordPress system administrators
- DevOps engineers
- Site reliability engineers (SREs)
- Technical support staff
- Security operations center (SOC) personnel

1.3 Conventions

Convention	Meaning
`code`	Commands, file paths, or configuration values
`[CUSTOMIZE: ...]`	Site-specific values to be filled in
> **WARNING:**	Critical safety information
> **NOTE:**	Important context or clarification
> **CUSTOMIZE:**	Settings specific to your environment
Expected:	Anticipated output or result
italics	Emphasis or variable placeholders
Procedure Metadata	Required operational fields: `Owner`, `Last Tested`, `Review Cadence`, and `Last Drill Date` (for incident/disaster recovery procedures)

Standard Procedure Block (for operational procedures):

- **Purpose**
- **Prerequisites**
- **Commands**
- **Expected Output**

- **Rollback**
- **Verification**
- **Escalate If**

1.4 Version History

Version	Date	Changes	Author
2.0	Feb 2026	Comprehensive update with security hardening sections	[CUSTOMIZE]
1.5	Aug 2025	Added disaster recovery procedures	[CUSTOMIZE]
1.0	Feb 2025	Initial release	[CUSTOMIZE]

1.5 Freshness Guardrails

Use the following controls to keep this runbook operationally reliable:

1. Every critical procedure listed in Appendix E must have an assigned owner.
2. `Last Tested` dates must be updated after execution in staging or production.
3. `Review Cadence` is mandatory; stale procedures are considered non-compliant.
4. Incident and disaster recovery procedures must include a `Last Drill Date`.
5. If metadata is stale, escalate to the runbook owner before the next change window.

Section 2: Site Overview

2.1 Site Identity

Property	Value
Domain	[CUSTOMIZE: example.com]
Site Name	[CUSTOMIZE: My WordPress Site]
WordPress Version	[CUSTOMIZE: current stable release, e.g. WordPress 6.9.1]

Property	Value
PHP Version	[CUSTOMIZE: 8.3+ recommended; test 8.4 in staging first]
Server OS	[CUSTOMIZE: Ubuntu 22.04 LTS]
Hosting Type	[CUSTOMIZE: Self-hosted/Managed/VPS]
Server Hostname	[CUSTOMIZE: wp-prod-01.example.com]
IP Address	[CUSTOMIZE: 192.168.1.100]
Database Engine	[CUSTOMIZE: MySQL 8.0 / MariaDB 10.6]

2.2 Credentials and Secrets

> **WARNING:** Never commit credentials to version control. Store all secrets in a secure vault (1Password, Vault, etc.).

Credential Type	Storage Location	Rotation Schedule	Owner
Database credentials	[CUSTOMIZE: Secure vault]	Every 90 days	[CUSTOMIZE]
SFTP/SSH keys	[CUSTOMIZE: Secure vault]	Every 180 days	[CUSTOMIZE]
Application passwords (if used)	[CUSTOMIZE: Secure vault + owner inventory]	Review quarterly; revoke on incident	[CUSTOMIZE]
Third-party API keys	[CUSTOMIZE: wp-config.php env vars]	Every 180 days	[CUSTOMIZE]
SSL certificates	[CUSTOMIZE: /etc/letsencrypt]	Auto-renew 30 days before expiry	[CUSTOMIZE]

2.3 User Roles and Responsibilities

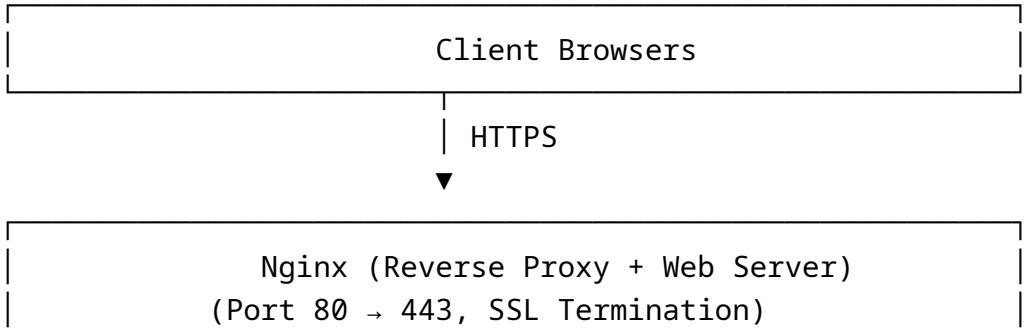
Role	Responsibilities	On-Call	Contact
System Administrator	Server infrastructure, backups, security	Yes	[CUSTOMIZE]
WordPress Administrator	Plugin management, user access, content	Yes	[CUSTOMIZE]
Database Administrator	Database maintenance, optimization, recovery	On-rotation	[CUSTOMIZE]
Security Officer	Vulnerability patching, security audits	On-rotation	[CUSTOMIZE]
Content Manager	Publishing, editorial workflow, media	Business hours	[CUSTOMIZE]

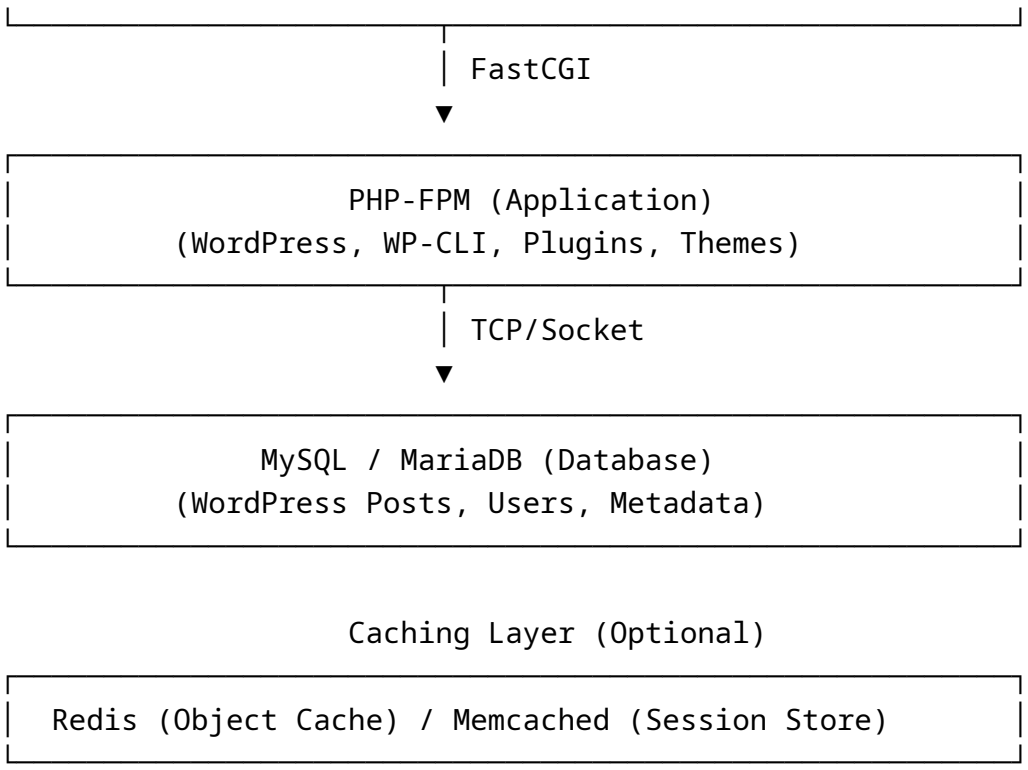
Section 3: Environment Reference

Unless a procedure explicitly says otherwise, host-level commands in this runbook assume a self-managed Linux deployment with shell access, direct service control, local backup paths, and direct database administration. On managed hosting platforms, use the provider's equivalent controls or provider-escalation path instead of forcing self-managed commands onto an environment that does not expose them.

3.1 Architecture Overview: LEMP Stack

The WordPress site runs on a LEMP (Linux, Nginx, MySQL, PHP) stack:





3.2 Service Configuration Reference

Service	Version	Port	User	Config Path	Log Path
Nginx	[CUSTOMIZE: 1.24+]	80, 443	`www-data`	`/etc/nginx/nginx.conf`	`/var/log/nginx/`
PHP-FPM	[CUSTOMIZE: 8.3+]	9000 (socket)	`www-data`	`/etc/php/[CUSTOM 8.x]/fpm/php.ini`	`/var/log/php*.log`
MySQL	[CUSTOMIZE: 8.0+]	3306 (local)	`mysql`	`/etc/mysql/mysql.conf.d/mysqld.cnf`	`/var/log/mysql/error.l`
Redis	[CUSTOMIZE: 7.0+]	6379	`redis`	`/etc/redis/redis`	`/var/log/redis/redis-server.log`

3.3 Caching Architecture

Layer	Technology	Purpose	TTL
Page Cache	[CUSTOMIZE: W3 Total Cache / WP Super Cache / Litespeed]	Full page HTML	1 hour
Object Cache	[CUSTOMIZE: Redis / Memcached / Object Cache Pro]	Database queries, transients	24 hours
Browser Cache	HTTP headers (Cache-Control)	Client-side assets	1 week (CSS/JS), 1 month (images)
Database Query Cache	MariaDB Query Cache (removed in MySQL 8.0+)	SELECT statements	Disabled / configured
CDN Cache	[CUSTOMIZE: Cloudflare / Bunny / AWS CloudFront]	Static assets, images	Varies

3.4 File System Locations

```
/home/wordpress/
├── public_html/                    # WordPress root
│   ├── wp-admin/                   # WordPress admin
│   ├── wp-content/
│   │   ├── plugins/                # Custom and third-party plugins
│   │   ├── themes/                 # Active and inactive themes
│   │   ├── uploads/                # User-uploaded media
│   │   │   ├── [YEAR]/[MONTH]/    # Organized by date
│   │   │   └── cache/              # Cache plugin files
│   │   └── mu-plugins/             # Must-use plugins
│   ├── wp-includes/                # WordPress core files
│   ├── wp-config.php               # Core configuration (SECRET)
│   ├── .htaccess                   # Rewrite rules (if using Apache)
│   ├── index.php                   # WordPress index
│   └── readme.html              # WordPress info (REMOVE IN PRODUCTION)
├── backup/                         # Local backup directory
│   ├── daily/
│   ├── weekly/
│   └── monthly/
├── logs/                           # Application logs
```

```
│   ├── access.log
│   ├── error.log
│   └── php-errors.log
└── ssl/                              # SSL certificates (optional)
    ├── cert.pem
    └── key.pem
```

3.5 SSL/TLS Configuration

> **CUSTOMIZE:** Update paths and domains based on your certificate provider.

Certificate Details:

Property	Value
Provider	[CUSTOMIZE: Let's Encrypt / Paid CA]
Certificate Path	[CUSTOMIZE: /etc/letsencrypt/live/example.com/]
Key Path	[CUSTOMIZE: /etc/letsencrypt/live/example.com/privkey.pem]
Certificate Chain	[CUSTOMIZE: /etc/letsencrypt/live/example.com/fullchain.pem]
Renewal Method	[CUSTOMIZE: Certbot cron / Manual renewal]
Renewal Schedule	30 days before expiry
Current Expiration	[CUSTOMIZE: YYYY-MM-DD]

Verify SSL Certificate:

```
# Check certificate details
openssl x509 -in /etc/letsencrypt/live/[CUSTOMIZE: example.com]/cert.pem -
noout -text

# Check certificate expiration
openssl x509 -in /etc/letsencrypt/live/[CUSTOMIZE: example.com]/cert.pem -
noout -dates

# Test SSL configuration
```

```
openssl s_client -connect [CUSTOMIZE: example.com]:443 -tls1_2

# Verify from Nginx
curl -vI https://[CUSTOMIZE: example.com]
```

HSTS (HTTP Strict Transport Security):

The following header is configured in Nginx to enforce HTTPS:

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload
# Note: The preload directive submits the domain to browser HSTS preload lists,
# which is difficult to reverse. Add preload only after confirming all subdomains
# support HTTPS and obtaining organizational approval.
```

This tells browsers to only connect via HTTPS for one year.

Section 4: Development Workflow

4.1 Version Control Strategy

Repository Structure:

```
wordpress-repo/
├── wp-content/
│   ├── plugins/            # Custom plugins
│   ├── themes/             # Custom themes
│   └── .gitignore
├── wp-config.php           # Template (no secrets)
├── .htaccess               # Rewrite rules
├── README.md               # Documentation
└── .git
```

> **WARNING:** Never commit to production directly. Always use the staging production workflow.

Branches:

- `main` - Production code (tagged releases)
- `develop` - Integration branch for features
- `hotfix/` - Emergency fixes for production
- `feature/` - New features for next release

4.2 Deployment Workflow

Time Estimate: 15-45 minutes depending on scope

Prerequisites: - All code reviewed and tested - Database backups current - Staging environment synchronized - Change window approved

Steps:

1. **Prepare Release Branch**

- `git checkout -b release/version-X.Y.Z develop`
  `git tag -a vX.Y.Z -m "Release version X.Y.Z"`

2. **Deploy to Staging First**

- `# SSH to staging server`
  `ssh [CUSTOMIZE: staging-user@staging.example.com]`
  `cd /home/wordpress/public_html`

  `# Pull latest code`
  `git fetch origin`
  `git checkout release/version-X.Y.Z`

  `# Install/update dependencies`
  `composer install --no-dev`

  `# Do not introduce live plugin/theme version drift during deployment.`
  `# Patch plugins/themes through the dedicated update workflow instead.`

  `# Run database migrations if needed`
  `wp db query < migrations/pending-migration.sql`

3. **Test on Staging**

    - Verify site loads: `curl -I https://[CUSTOMIZE: staging.example.com]`
    - Check critical pages for functionality
    - Run automated tests: `npm run test`
    - Verify no console errors
    - Test user registration/login
    - Test form submissions

4. **Approve for Production**

    - Notify team lead
    - Get sign-off from project manager

- Document any issues found

5. **Deploy to Production**

- ```
  # SSH to production server
  ssh [CUSTOMIZE: prod-user@prod.example.com]
  cd /home/wordpress/public_html

  # Create backup before deploy
  TS=$(date +%Y%m%d-%H%M%S)
  wp db export /home/wordpress/backup/pre-deploy-${TS}.sql
  gzip /home/wordpress/backup/pre-deploy-${TS}.sql

  # Pull latest code
  git fetch origin
  git checkout release/version-X.Y.Z

  # Install dependencies
  composer install --no-dev

  # Do not run live plugin/theme updates during deployment.
  # Deploy only the approved release artifact.
  ```

6. **Post-Deployment Verification**

- ```
  # Verify WordPress is healthy
  wp core is-installed && echo "WordPress OK"

  # Check for PHP errors in logs
  tail -20 /var/log/php-errors.log

  # Verify database connection
  wp db check

  # Test site responsiveness
  curl -s -w "%{http_code}\n" -o /dev/null https://[CUSTOMIZE: example.com]
  ```

7. **Monitor for Issues**

   - Watch error logs for 15 minutes
   - Monitor site uptime and response time
   - Check critical user flows
   - Be ready to rollback if needed

4.3 Staging Synchronization

Sync Staging Environment from Production

Time Estimate: 10-30 minutes

> **WARNING:** Staging sync will overwrite staging data. Notify team before starting.

1. **Backup Staging Database (in case of error)**

- ```
  ssh [CUSTOMIZE: staging-user@staging.example.com]
  cd /home/wordpress
  wp db export /home/wordpress/backup/staging-pre-sync-$(date +%Y%m%d-%H%M%S).sql
  ```

2. **Export Production Database**

- ```
  ssh [CUSTOMIZE: prod-user@prod.example.com]
  cd /home/wordpress
  wp db export - --single-transaction > /tmp/prod-dump.sql
  ```

3. **Transfer to Staging**

- ```
  # From staging server
  scp [CUSTOMIZE: prod-user@prod.example.com]:/tmp/prod-dump.sql ./prod-dump.sql
  ```

4. **Import to Staging**

- ```
  # On staging server
  # WARNING: This overwrites the staging database with production data.
  wp db import prod-dump.sql

  # WARNING: This modifies all URL references across all database tables.
  wp search-replace "https://example.com" "https://staging.example.com" --all-tables
  ```

5. **Sync Code**

- ```
  # On staging server
  git fetch origin
  git checkout origin/main
  wp plugin list
  ```

6. **Clear Caches**

- ```
  wp cache flush
  # Plugin-dependent — uncomment the cache plugin(s) in use:
  ```

```
    # wp w3-total-cache flush all
    # wp redis flush-db
```

Section 5: Security Hardening

5.1 Firewall and SSH Configuration

UFW (Uncomplicated Firewall) Setup:

```
# Enable firewall
sudo ufw enable

# Allow SSH (critical - do this FIRST)
sudo ufw allow 22/tcp

# Allow HTTP and HTTPS
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Block all other incoming traffic
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Verify rules
sudo ufw status
```

Expected: SSH, HTTP, HTTPS allowed; all other ports denied.

SSH Hardening (in `/etc/ssh/sshd_config`):

```
# Disable root login
PermitRootLogin no

# Disable password authentication (use keys only)
PasswordAuthentication no
PubkeyAuthentication yes

# Change default port (optional - use non-standard port)
Port [CUSTOMIZE: 2222]
```

```
# Limit concurrent sessions
MaxStartups 10:30:100

# Set login grace time
LoginGraceTime 30s

# Enable X11 forwarding only if needed
X11Forwarding no

# Restrict users who can SSH
AllowUsers [CUSTOMIZE: admin@*.example.com wordpress@*.example.com]
```

Restart SSH:

```
sudo systemctl restart sshd
```

> **WARNING:** Test SSH access before closing your current session to avoid lockout.

5.2 WordPress Hardening

Remove Unnecessary Public Metadata Files:

```
# Remove readme files
rm /home/wordpress/public_html/readme.html
rm /home/wordpress/public_html/wp-admin/readme.html

# Remove license files
rm /home/wordpress/public_html/license.txt
```

Do not rely on manually deleting `wp-admin/install.php` as a hardening control. WordPress blocks installer reuse after setup, and core files may be restored during updates.

Disable File Editing:

Add to `wp-config.php`:

```
// Baseline: disable the built-in theme/plugin editor
define('DISALLOW_FILE_EDIT', true);

// Optional hardened profile: disable all Dashboard file modifications
// define('DISALLOW_FILE_MODS', true);
```

> **WARNING:** If you enable `DISALLOW_FILE_MODS`, Dashboard-based plugin/theme installs and updates are disabled. Use only with a documented deployment/update pipeline.

Disable XML-RPC (Recommended):

Block at the web server level (preferred). See Section 5.4 for details.

Restrict REST API Access (Optional):

See Section 5.4 for REST API management and the WordPress Security Hardening Guide §7.5 for comprehensive REST API security guidance.

5.3 HTTP Security Headers

Configure in Nginx (`/etc/nginx/conf.d/security-headers.conf`):

```
# Prevent MIME type sniffing
add_header X-Content-Type-Options "nosniff" always;

# Clickjacking protection - prevent embedding in frames
add_header X-Frame-Options "SAMEORIGIN" always;

# Referrer Policy
add_header Referrer-Policy "strict-origin-when-cross-origin" always;

# Permissions Policy (formerly Feature Policy) - limit browser features
add_header Permissions-Policy "geolocation=(), microphone=(), camera=()" always;

# Content Security Policy (CSP) - restrict content sources
# WARNING: unsafe-eval weakens CSP significantly. Remove where possible.
# See Benchmark 1.2 for hardened CSP guidance.
add_header Content-Security-Policy "default-src 'self'; script-
src 'self' 'unsafe-inline' 'unsafe-eval' cdn.example.com; style-
src 'self' 'unsafe-inline'; img-src 'self' data: https:; font-
src 'self' data:;" always;

# HSTS - Force HTTPS
# Note: Add preload only after confirming all subdomains support HTTPS
# and obtaining organizational approval.
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preloa
```

Reload Nginx:

```
sudo nginx -t  # Test configuration
sudo systemctl reload nginx
```

Verify Headers:

```
curl -I https://[CUSTOMIZE: example.com] | grep -i "X-\|Strict\|Content-Security"
```

> **NOTE:** See WordPress Security Benchmark §1.2 for the complete audit procedure, rationale for each header, and Level 2 guidance on removing `unsafe-inline` from CSP.

5.4 REST API and XML-RPC Management

> **NOTE:** For comprehensive REST API security guidance (authentication methods, permission callbacks, CORS, data validation), see the WordPress Security Hardening Guide §7.5. For prescriptive benchmarks on rate limiting, unauthenticated access, XML-RPC disablement, and user enumeration prevention, see WordPress Security Benchmark §1.5, §4.4, §5.4, and §5.6.

Disable XML-RPC (recommended):

Option 1 (preferred): Block at the web server level in Nginx `/etc/nginx/conf.d/block-xmlrpc.conf`:

```
location = /xmlrpc.php {
    deny all;
}
```

Option 2: Disable via a must-use plugin (`wp-content/mu-plugins/disable-xmlrpc.php`):

```
<?php
add_filter( 'xmlrpc_enabled', '__return_false' );
```

Additionally, disable trackbacks and pingbacks in **Settings** → **Discussion** by unchecking "Allow link notifications from other blogs (pingbacks and trackbacks) on new posts."

Scope REST API Exposure (Recommended):

Keep required public endpoints available (for example, posts on public sites) and restrict only sensitive routes.

Example must-use plugin (`wp-content/mu-plugins/restrict-rest-users.php`) to restrict user-enumeration to authorized users (see Benchmark 5.6):

```
<?php
// Restrict /wp/v2/users to users with the list_users capability.
add_filter( 'rest_endpoints', function( $endpoints ) {
    if ( isset( $endpoints['/wp/v2/users'] ) ) {
        foreach ( $endpoints['/wp/v2/users'] as $i => $route ) {
        $endpoints['/wp/v2/users'][ $i ]['permission_callback'] = function() {
                return current_user_can( 'list_users' );
            };
        }
    }
    if ( isset( $endpoints['/wp/v2/users/(?P<id>[\\d]+)'] ) ) {
      foreach ( $endpoints['/wp/v2/users/(?P<id>[\\d]+)'] as $i => $route ) {
        $endpoints['/wp/v2/users/(?P<id>[\\d]+)'][ $i ]['permission_callback'] = f
                return current_user_can( 'list_users' );
            };
        }
    }
    return $endpoints;
} );
```

> **WARNING:** Avoid blanket REST API authentication requirements unless architecture explicitly requires it. Global blocking often breaks headless front ends, plugins, and theme features.

Monitor REST API Usage:

```
# Check Nginx logs for REST API calls
grep "wp-json" /var/log/nginx/access.log | tail -20

# Check for high-volume API callers
awk '/wp-json/ {print $1}' /var/log/nginx/access.log | sort | uniq -
c | sort -rn | head -20
```

5.5 Administrator Two-Factor Authentication (2FA)

Objective: Require 2FA for all privileged accounts and maintain a documented break-glass recovery path.

See WordPress Security Benchmark §5.1 for the compliance audit checklist and configuration rationale.

Choose and Standardize One 2FA Plugin:

- `two-factor`.

Implementation Steps:

1. Install and activate the approved 2FA plugin:

```
wp plugin install two-factor --activate
```

2. In plugin settings, enforce 2FA for all privileged roles:
    - single-site: `administrator`, `editor`, and any additional role with equivalent authority (for example, `shop_manager`)
    - Multisite: all `Super Admin` accounts plus any site-level admin roles that require elevated access
3. Enroll all privileged accounts and store backup recovery codes in the approved password vault.
4. Verify operational state:

```
wp plugin is-active two-factor && echo "2FA plugin active"
wp user list --role=administrator --fields=ID,user_login,user_email --format=table
wp user list --role=editor --fields=ID,user_login,user_email --format=table

# Multisite only
wp super-admin list
```

5. Document exceptions and break-glass approvals (owner, approver, expiry, and rollback steps).

Emergency Recovery (temporary):

```
# Preferred: recover the affected account without disabling the plugin globally.
# Use action-gated reauthentication or equivalent identity re-verification first.

# Last resort only during approved incident response:
wp plugin deactivate two-factor

# Re-enable immediately after account recovery
wp plugin activate two-factor
```

> **WARNING:** Any bypass window must be ticketed, time-bounded, approved by the incident owner, and reviewed after closure. When a global disable is unavoidable, apply compensating controls immediately and re-enroll affected users before closing the incident.

Privileged Action Reauthentication (Sudo Mode) **Purpose:** Require action-gated reauthentication before sensitive or destructive operations even when the operator is already logged in. Managed platforms may call this *step-up authentication*; for example, WordPress VIP uses that term for higher-risk Dashboard actions.

See WordPress Security Benchmark §5.5 and WordPress Security Hardening Guide §8.2 for the corresponding policy rationale.

Prerequisites:

- 2FA is active for all privileged accounts
- Approved reauthentication control selected (plugin, custom control, or managed-hosting equivalent)
- Documented exception and break-glass owner

Commands:

```
# Plugin-dependent - uncomment the approved reauthentication control in use:
# wp plugin is-active fortress && echo "Fortress active"
# wp plugin is-active wp-sudo && echo "wp-sudo active"

# Multisite only - verify current Super Admin inventory before gating network-level actions
wp super-admin list
```

Protected actions must include, at minimum:

- installing, activating, or deleting plugins and themes
- promoting users to privileged roles
- creating or revoking application passwords
- editing `wp-config.php` or equivalent critical configuration
- performing core upgrades or downgrades
- executing 2FA bypass or recovery steps
- exporting or restoring production data

Expected Output:

- The approved control is active.
- Sensitive actions trigger a fresh password and 2FA challenge before execution.

Rollback:

- If the control blocks emergency remediation, document the exception, apply the temporary bypass for the shortest practical window, complete the recovery action, and restore the gate before closing the incident.

Verification:

- Test at least one action from each category during staging or a quarterly drill.
- Confirm the challenge applies to the Dashboard and relevant API surfaces where those actions are exposed.

Escalate If:

- The reauthentication control blocks all privileged access without a documented recovery path.
- A sensitive action can still be completed without a fresh challenge.
- Multisite network actions are not covered for `Super Admin` accounts.

5.6 File Integrity Monitoring (FIM)

Setup AIDE (Advanced Intrusion Detection Environment):

```
# Install AIDE
sudo apt-get install aide aide-common

# Initialize AIDE database (takes several minutes)
sudo aideinit

# Move database to production location
sudo mv /var/lib/aide/aide.db /var/lib/aide/aide.db.orig
sudo cp /var/lib/aide/aide.db.new /var/lib/aide/aide.db

# Check file integrity
sudo aide --check

# Update database after planned changes
sudo aide --init
sudo mv /var/lib/aide/aide.db.new /var/lib/aide/aide.db
```

Configure Automated Checking:

Add to crontab (`crontab -e`):

```
# Run AIDE check daily at 2 AM
0 2 * * * /usr/bin/aide --check | mail -s "AIDE Report for $(hostname)" root@example.
```

Monitor WordPress File Changes:

```
# List recently modified WordPress files (potential hack indicator)
find /home/wordpress/public_html -type f -mtime -1 -ls

# Check for suspicious file permissions
find /home/wordpress/public_html -type f -perm /022 -ls

# Find recently added PHP files
find /home/wordpress/public_html -name "*.php" -type f -mtime -7
```

Section 6: Routine Maintenance

Lifecycle metadata for critical maintenance procedures is tracked in Appendix E.

6.1 Maintenance Calendar

Task	Frequency	Duration	Owner	Window
Monitor error logs	Daily	5 min	Sysadmin	Any time
Check plugin/theme updates	Daily	10 min	Sysadmin	8 AM
Review up-time/performance	Daily	5 min	Sysadmin	9 AM
Database optimization	Weekly	15 min	DBA	Sun 2 AM
WordPress core updates	Weekly*	30 min	Sysadmin	Scheduled
Plugin/theme updates	Weekly*	45 min	Sysadmin	Scheduled

Task	Frequency	Duration	Owner	Window
Backup verification	Weekly	20 min	Sysadmin	Mon 6 AM
Security patch review	Weekly	30 min	Security	Any time
PHP version check	Monthly	1 hour	Sysadmin	Scheduled
Log rotation check	Monthly	10 min	Sysadmin	1st Sun
Firewall rule audit	Monthly	20 min	Sysadmin	1st Sun
Full backup check	Monthly	1 hour	DBA	Last Sun

* *Update as available; prioritize security updates immediately*

6.2 WordPress Core Updates

Time Estimate: 30-45 minutes

Purpose: Apply WordPress core updates with a validated backup and post-update checks, minimizing downtime and regression risk.

See WordPress Security Benchmark §4.5 for the configuration audit and rationale behind automatic core update settings.

Prerequisites: - Database backup created - No deploys in progress - Monitor available for next 30 minutes

Commands:

1. **Check Current Version and Available Updates**

- ```
  wp core version
  wp core check-update
  ```

2. **Create Pre-Update Database Backup**

- ```
  TS=$(date +%Y%m%d-%H%M%S)
  wp db export /home/wordpress/backup/pre-core-update-${TS}.sql
  gzip /home/wordpress/backup/pre-core-update-${TS}.sql
  ```

3. **Update WordPress Core and Database Schema**

- `wp core update`
  `wp core update-db`

4. **Flush Runtime Caches**

- `wp cache flush`
  `# Plugin-dependent — uncomment the cache plugin(s) in use:`
  `# wp w3-total-cache flush all`
  `# wp redis flush-db`

Expected Output:

- `wp core check-update` returns no pending core updates.
- `wp core version` reports the intended target version.
- No new fatal errors in PHP, web server, or debug logs.

Rollback:

```
# Restore the pre-update database backup
gunzip < /home/wordpress/backup/pre-core-update-YYYYMMDD-HHMMSS.sql.gz | wp db imp
```

```
# Restore a known-good core/files artifact if the update changed runtime-
managed files
tar -xzf /home/wordpress/backup/wordpress_TIMESTAMP.tar.gz -C /
```

Use Section 8.3 for full rollback workflow.

Verification:

```
# Confirm new version
wp core version

# Check for errors in logs
tail -20 /var/log/php-errors.log
tail -20 wp-content/debug.log

# Verify site response
curl -I https://[CUSTOMIZE: example.com]
```

Then verify Dashboard login and critical user flows in browser.

Escalate If:

- Site returns non-2xx/3xx after update.

- Fatal errors persist after cache flush and rollback.
- Database upgrade (`wp core update-db`) fails.

> **WARNING:** Minor updates are generally lower risk. Major updates should be tested on staging first and aligned with the current supported major release.

6.3 Plugin and Theme Updates

Time Estimate: 45-90 minutes depending on number of updates

Purpose: Apply plugin and theme updates with controlled blast radius, rapid detection of regressions, and a clear rollback path.

The WordPress Security Benchmark §8.3 defines the patching SLA: security updates within 72 hours, critical patches within 24 hours or virtual-patched immediately.

Prerequisites: - All updates tested on staging - Database backed up - Monitor available during and after

Commands:

1. **List Available Updates**

- ```
  wp plugin list --format=table --fields=name,status,update
  wp theme list --format=table --fields=name,status,update
  ```

2. **Create Pre-Update Database Backup**

- ```
  TS=$(date +%Y%m%d-%H%M%S)
  wp db export /home/wordpress/backup/pre-plugin-update-${TS}.sql
  gzip /home/wordpress/backup/pre-plugin-update-${TS}.sql
  ```

3. **Update Plugins in Controlled Order**

- ```
  # Update one plugin at a time, then test before proceeding
  wp plugin update plugin-name

  # Spot-check logs after each update
  tail -10 /var/log/php-errors.log

  # Verify site returns a healthy status
  curl -so /dev/null -w "%{http_code}" https://[CUSTOMIZE: example.com]
  ```

4. **Update Themes**

- ```
  # If using child theme, update parent only
  wp theme update parent-theme-name
  ```

5. **Confirm Active Plugin State**

- `wp plugin list --status=active --fields=name,status,version --format=table`

  ```
  # Reactivate only if a plugin deactivated unexpectedly
  wp plugin activate plugin-name
  ```

6. **Post-Update Checks**

- ```
  # Verify WordPress core data
  wp db check

  # Clear caches
  wp cache flush

  # Test critical pages
  curl -s https://[CUSTOMIZE: example.com] | grep "<title>"
  ```

Expected Output:

- Updated components show `update: none`.
- Site remains reachable (`200/301/302`) during update window.
- No new fatal errors in `/var/log/php-errors.log` or `wp-content/debug.log`.

Rollback:

```
# Restore database if needed
gunzip < /home/wordpress/backup/pre-plugin-update-YYYYMMDD-HHMMSS.sql.gz | wp db i


# Restore the known-good filesystem artifact for the affected plugin/theme set
tar -xzf /home/wordpress/backup/wordpress_TIMESTAMP.tar.gz -C /
```

Use Section 8.3 for full rollback steps.

Verification:

- Validate login, checkout/forms, search, and caching behavior.
- Confirm plugin/theme health in Dashboard (no fatal notices or forced deactivations).

Escalate If:

- Multiple plugins fail update or auto-deactivate.
- Critical user workflows fail after rollback attempt.
- Repeated fatal errors persist after isolating updated components.
```

> **NOTE:** Keep a log of updates in your change management system. Document any issues encountered.

6.4 Database Optimization

Time Estimate: 15-30 minutes

Optimize Database (Non-Destructive):

```
# Check database size
wp db query "SELECT ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS 'Datab

# Optimize all tables
wp db optimize

# Check for database errors
wp db check

# If errors found, repair:
wp db repair

# View database statistics
wp db query "SHOW TABLE STATUS;"
```

Remove Old Revisions:

```
# Count revisions
wp db query "SELECT COUNT(*) AS revision_count FROM $(wp db prefix)posts WHERE post_

# WARNING: This permanently deletes all post revisions. They cannot be recovered.
wp post list --post_type=revision --format=ids | xargs wp post delete -
-force
```

Or add to `wp-config.php` to limit future revisions:

```
define('WP_POST_REVISIONS', 5);
```

Clean Up Spam Comments:

```
# List spam comments
wp comment list --status=spam --format=table --fields=comment_ID,comment_author,co
```

```
# WARNING: This permanently deletes all spam comments. They cannot be recovered.
wp comment delete $(wp comment list --status=spam --format=ids) --force
```

Remove Unused Transients:

```
# Delete expired transients
wp transient delete --expired
```

6.5 PHP Version Upgrade

Time Estimate: 1-2 hours including testing

> **WARNING:** Major PHP version changes can break plugins/themes. Always test on staging first.
>
> **NOTE:** Replace 8 . 3 below with your target PHP version (for example, 8 . 4 after staging validation) and keep package names, FPM service, pool path, and socket references consistent.

Prerequisites: - All plugins compatible with new PHP version (check WordPress.org) - All themes compatible - Staging environment updated first - Database backed up - 2-hour maintenance window available

Steps:

1. **Check Current PHP Version**

- ```
  php -v
  wp core version
  wp plugin list --fields=name,status,version,update --format=table
  # Verify PHP compatibility for critical plugins in vendor docs before upgrade.
  ```

2. **Update PHP on Staging**

- ```
  # Update PHP (example for Ubuntu/Debian)
  sudo apt-get update
  sudo apt-get install -y php8.3 php8.3-fpm php8.3-mysql php8.3-
  gd php8.3-xml php8.3-curl php8.3-mbstring

  # Switch to new PHP version
  sudo update-alternatives --set php /usr/bin/php8.3
  ```

3. **Update PHP-FPM Pool Configuration**

- ```
  # Edit pool configuration
  sudo nano /etc/php/8.3/fpm/pool.d/www.conf

  # Restart PHP-FPM
  sudo systemctl restart php8.3-fpm
  ```

4. **Test on Staging**

- ```
  # Verify site works
  curl -I https://[CUSTOMIZE: staging.example.com]

  # Check for PHP errors
  tail -30 /var/log/php8.3-fpm.log

  # Run automated tests
  npm run test
  ```

5. **Update Nginx Configuration** (if needed)

- ```
  # In /etc/nginx/sites-available/default
  upstream php {
      server unix:/run/php/php8.3-fpm.sock;  # Update version
  }
  ```

6. **Deploy to Production**

- ```
  # During maintenance window
  sudo apt-get install -y php8.3-fpm php8.3-mysql php8.3-gd php8.3-
  xml php8.3-curl php8.3-mbstring

  # Update Nginx
  sudo nginx -t
  sudo systemctl reload nginx

  # Restart PHP-FPM
  sudo systemctl restart php8.3-fpm

  # Verify
  curl -I https://[CUSTOMIZE: example.com]
  ```

7. **Monitor for Issues**

    - Watch error logs for 30 minutes
    - Test critical functionality
    - Verify form submissions work
    - Check database connections

6.6 WordPress Cron (WP-Cron) Management

Purpose: Ensure scheduled WordPress tasks run predictably by using system cron with WP-CLI, while reducing external exposure of `wp-cron.php`.

WP-Cron uses WordPress hooks to trigger background tasks when site traffic occurs. It is not a true system scheduler.

> **NOTE:** If your site has low traffic, WP-Cron tasks may not run on schedule. Consider using system cron instead.

Prerequisites:

- Shell access to the web server host
- WP-CLI functional in the WordPress root
- Change window for `wp-config.php` and web server config updates

Commands:

```
# List all scheduled cron events
wp cron event list

# Run due events immediately (sanity check)
wp cron event run --due-now
```

Add to `wp-config.php`:

```
define('DISABLE_WP_CRON', true);
```

Install a system cron entry:

```
# Add once; ensure duplicates are not introduced
(crontab -l 2>/dev/null; echo "*/5 * * * * cd /home/wordpress/public_html && wp cron
-due-now > /dev/null 2>&1") | crontab -
```

Once system cron is configured, block external access to `wp-cron.php` at the web server level:

```
location = /wp-cron.php {
    deny all;
}
```

Expected Output:

- `crontab -l` shows one `wp cron event run --due-now` entry.
- `wp cron event run --due-now` runs without fatal errors.
- External `GET /wp-cron.php` returns `403` after web server reload.

Rollback:

```
# Remove system cron entry
crontab -l | grep -v "wp cron event run --due-now" | crontab -
```

Then remove `define('DISABLE_WP_CRON', true);` from `wp-config.php` and restore previous web server config.

Verification:

```
crontab -l | grep "wp cron event run --due-now"
wp cron event list
curl -s -o /dev/null -w "%{http_code}\n" https://[CUSTOMIZE: example.com]/wp-cron.php
```

Escalate If:

- Scheduled tasks stop running after migration to system cron.
- WP-CLI cron command fails repeatedly.
- `wp-cron.php` cannot be reliably restricted at the web server layer.

> **NOTE:** Do not use `curl` to hit `wp-cron.php` over HTTP as the system cron replacement. The `wp-cron.php` endpoint should be blocked to prevent resource exhaustion attacks. Use WP-CLI directly instead. See WordPress Security Benchmark §4.7 for the full rationale.

6.7 Transactional Email Management

Email Delivery Configuration:

WordPress should send emails through an SMTP service, not PHP mail():

Using WP Mail SMTP Plugin:

```
# Install and activate
wp plugin install wp-mail-smtp --activate
```

Configure the plugin in the Dashboard or lock the settings with the plugin's documented constants. Do not attempt to update unsupported per-field SMTP options with generic `wp option update` commands.

Example Constant-Based Configuration for WP Mail SMTP:

```
// WP Mail SMTP plugin constants
define('WPMS_ON', true);
define('WPMS_MAILER', 'smtp');
define('WPMS_SMTP_HOST', '[CUSTOMIZE: smtp.sendgrid.net]');
define('WPMS_SMTP_PORT', 587);
define('WPMS_SSL', 'tls');
define('WPMS_SMTP_AUTH', true);
define('WPMS_SMTP_USER', '[CUSTOMIZE: apikey]');
define('WPMS_SMTP_PASS', '[CUSTOMIZE: sg.xxx...]');
define('WPMS_FROM_EMAIL', 'noreply@[CUSTOMIZE: example.com]');
// Optional:
// define('WPMS_FROM_NAME', '[CUSTOMIZE: Site Name]');
```

Test Email Delivery:

```
# Send test email
wp eval-file - <<'PHP'
wp_mail('[CUSTOMIZE: admin@example.com]', 'WordPress Email Test', 'This is a test e
PHP

# Check for errors
tail -20 /var/log/php-errors.log
```

6.8 Log Rotation

Configure Log Rotation (`/etc/logrotate.d/wordpress`):

```
/var/log/nginx/*.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 www-data adm
    sharedscripts
    postrotate
        if [ -f /run/nginx.pid ]; then
            kill -USR1 $(cat /run/nginx.pid)
        fi
    endscript
```

```
}

/home/wordpress/logs/*.log {
    daily
    missingok
    rotate 30
    compress
    delaycompress
    notifempty
    create 0640 www-data www-data
}

/var/log/php*.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    sharedscripts
    postrotate
        /usr/lib/php/php-fpm-checkconf > /dev/null 2>&1 || true
        if [ $? -eq 0 ]; then
        systemctl reload [CUSTOMIZE: php_fpm_service] > /dev/null 2>&1 || true
        fi
    endscript
}

/var/log/mysql/error.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    create 640 mysql mysql
}
```

Test Log Rotation:

```
# Test logrotate configuration
```

```
sudo logrotate -d /etc/logrotate.d/wordpress

# Force immediate rotation (testing only)
sudo logrotate -f /etc/logrotate.d/wordpress

# Verify rotation worked
ls -lah /var/log/nginx/*.log*
```

6.9 Monitoring and Alerting

Key Metrics to Monitor:

Metric	Threshold	Check Frequency
Uptime	99.9%	Every 5 minutes
Response Time	< 2 seconds	Every 60 seconds
CPU Usage	< 80%	Every 60 seconds
Memory Usage	< 90%	Every 60 seconds
Disk Space	> 20% free	Every 60 seconds
Error Rate	< 1%	Every 60 seconds

Using Prometheus + Grafana (Optional):

```
# Install node_exporter for server metrics
wget https://github.com/prometheus/node_exporter/releases/download/v1.6.0/node_ex
1.6.0.linux-amd64.tar.gz
tar xvfz node_exporter-1.6.0.linux-amd64.tar.gz
sudo mv node_exporter-1.6.0.linux-amd64/node_exporter /usr/local/bin/

# Create systemd service
sudo tee /etc/systemd/system/node_exporter.service > /dev/null <<EOF
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=prometheus
ExecStart=/usr/local/bin/node_exporter
```

```
[Install]
WantedBy=multi-user.target
EOF

sudo systemctl daemon-reload
sudo systemctl enable node_exporter
sudo systemctl start node_exporter
```

Simple Monitoring Script:

```
#!/bin/bash
# Check WordPress health every 5 minutes

SITE_URL="https://[CUSTOMIZE: example.com]"
RESPONSE=$(curl -s -w "%{http_code}" -o /dev/null $SITE_URL)

if [ "$RESPONSE" != "200" ]; then
      echo "ALERT: Site returned $RESPONSE at $(date)" | mail -
s "WordPress Site Down" root@example.com
    # Attempt to restart PHP-FPM
    systemctl restart [CUSTOMIZE: php_fpm_service]
fi

# Check database
if ! wp db check > /dev/null 2>&1; then
      echo "ALERT: Database check failed at $(date)" | mail -
s "WordPress DB Error" root@example.com
fi

# Check disk space
DISK_USAGE=$(df /home/wordpress | awk 'NR==2 {print $5}' | sed 's/%//')
if [ "$DISK_USAGE" -gt 90 ]; then
     echo "ALERT: Disk usage at ${DISK_USAGE}% at $(date)" | mail -
s "WordPress Disk Full" root@example.com
fi
```

Add to crontab:

```
*/5 * * * * /usr/local/bin/wordpress-health-check.sh
```

Section 7: Backup Procedures

7.1 Backup Strategy

Component	Frequency	Method	Retention	Storage
Database	Every 6 hours	`wp db export` (compressed) + optional physical backup	90 days	Encrypted offsite object storage or equivalent
WordPress Files	Daily	tar/rsync	30 days	Encrypted offsite object storage
Uploads Directory	Daily	tar/rsync	30 days	Encrypted offsite object storage
Configuration Files	Weekly	git + encrypted backup	1 year	Git + secure vault
Full Site Snapshot	Weekly	Volume or image snapshot	90 days	Encrypted offsite snapshot service

> **WARNING:** Test all backups quarterly. A backup that has never been restored is not a backup.
>
> **WARNING:** Offsite backup storage must be encrypted and isolated from normal production-host access. Use write-only or tightly scoped credentials where possible so the production host can upload new backups without being able to browse or delete retained recovery points.

7.2 Automated Backup Script

Create `/usr/local/bin/wordpress-backup.sh`:

```
#!/bin/bash
```

```
set -euo pipefail

WP_ROOT="/home/wordpress/public_html"
BACKUP_DIR="/home/wordpress/backup"
SITE_SLUG="[CUSTOMIZE: example.com]"
REMOTE_URI="s3://[CUSTOMIZE: backup-bucket]/${SITE_SLUG}/"
RETENTION_DAYS=90
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
LOG_FILE="/var/log/wordpress-backup.log"

# Create backup directory if it doesn't exist
mkdir -p "$BACKUP_DIR"
cd "$WP_ROOT"

echo "[$(date)] Starting WordPress backup..." >> "$LOG_FILE"

# 1. Database backup
echo "[$(date)] Backing up database..." >> "$LOG_FILE"
wp --path="$WP_ROOT" db export "$BACKUP_DIR/db_${TIMESTAMP}.sql" --
single-transaction >> "$LOG_FILE" 2>&1
gzip "$BACKUP_DIR/db_${TIMESTAMP}.sql"

# 2. WordPress files backup (excluding uploads and cache)
echo "[$(date)] Backing up WordPress files..." >> "$LOG_FILE"
tar --exclude='wp-content/uploads' --exclude='wp-content/cache' \
    --exclude='.git' --exclude='node_modules' \
    -czf "$BACKUP_DIR/wordpress_${TIMESTAMP}.tar.gz" \
    /home/wordpress/public_html >> "$LOG_FILE" 2>&1

# 3. Uploads directory backup
echo "[$(date)] Backing up uploads..." >> "$LOG_FILE"
tar -czf "$BACKUP_DIR/uploads_${TIMESTAMP}.tar.gz" \
    /home/wordpress/public_html/wp-content/uploads >> "$LOG_FILE" 2>&1

# 4. Upload to remote storage (example using AWS S3)
# Bucket policy and credentials should enforce encryption and prevent routine
# browse/delete access from the production host.
echo "[$(date)] Uploading to S3..." >> "$LOG_FILE"
aws s3 cp "$BACKUP_DIR/db_${TIMESTAMP}.sql.gz" "$REMOTE_URI" >> "$LOG_FILE" 2>&1
aws s3 cp "$BACKUP_DIR/wordpress_${TIMESTAMP}.tar.gz" "$REMOTE_URI" >> "$LOG_FILE"
aws s3 cp "$BACKUP_DIR/uploads_${TIMESTAMP}.tar.gz" "$REMOTE_URI" >> "$LOG_FILE" 2>
```

```
# 5. Cleanup old local backups
echo "[$(date)] Cleaning up old backups..." >> "$LOG_FILE"
find "$BACKUP_DIR" -type f -mtime +$RETENTION_DAYS -delete

# 6. Verify backup integrity
echo "[$(date)] Verifying backup..." >> "$LOG_FILE"
for file in "$BACKUP_DIR"/*_${TIMESTAMP}.*; do
    if [ -f "$file" ]; then
        SIZE=$(du -h "$file" | cut -f1)
     echo "[$(date)] Backup file: $(basename $file) (Size: $SIZE)" >> "$LOG_FILE"
    fi
done

echo "[$(date)] Backup completed successfully" >> "$LOG_FILE"
```

Make Script Executable:

```
sudo chmod +x /usr/local/bin/wordpress-backup.sh
sudo chown root:root /usr/local/bin/wordpress-backup.sh
```

Schedule with Cron:

```
# Run backup every 6 hours
0 */6 * * * /usr/local/bin/wordpress-backup.sh

# Run weekly full snapshot at 2 AM Sunday
0 2 * * 0 /usr/local/bin/wordpress-backup.sh
```

7.3 Backup Verification

Monthly Backup Test:

Time Estimate: 30-45 minutes

1. **List Available Backups**

- `ls -lh /home/wordpress/backup/ | tail -20`

  ```
  # Or check S3
  aws s3 ls s3://[CUSTOMIZE: backup-bucket]/[CUSTOMIZE: example.com]/ -
  -human-readable
  ```

2. **Verify Backup Integrity**

- ```
  # Test database export
  gunzip -t /home/wordpress/backup/db_*.sql.gz

  # Verify tar archives
  tar -tzf /home/wordpress/backup/wordpress_*.tar.gz | head -20
  tar -tzf /home/wordpress/backup/uploads_*.tar.gz | head -20
  ```

3. **Perform Test Restore (On Staging)**

- ```
  # Extract database backup
  zcat /home/wordpress/backup/db_TIMESTAMP.sql.gz > /tmp/test-restore.sql

  # Import to test database
  wp db import /tmp/test-restore.sql --allow-root

  # Verify tables
  wp db tables --all-tables --format=table
  ```

4. **Test File Restore**

- ```
  # Extract to temporary location
  mkdir -p /tmp/restore-test
  tar -xzf /home/wordpress/backup/wordpress_TIMESTAMP.tar.gz -C /tmp/restore-test

  # Verify key files exist
  ls -la /tmp/restore-test/home/wordpress/public_html/wp-config.php
  ls -la /tmp/restore-test/home/wordpress/public_html/index.php
  ```

5. **Document Results**

- ```
  Backup Verification Report - [DATE]

   Database backup: [SIZE], verified
   WordPress files backup: [SIZE], verified
   Uploads backup: [SIZE], verified
   Test restore successful
   All files present and readable

  Next verification: [DATE 30 days from now]
  ```

Section 8: Deployment Procedures

Lifecycle metadata for deployment procedures is tracked in Appendix E.

8.1 Code Deployment

Time Estimate: 15-30 minutes (excluding QA window)

Purpose: Deploy approved release code to production with post-deploy validation and fast rollback readiness.

See Section 4.2 for complete deployment workflow.

Prerequisites:

- Approved release/tag exists
- Deployment window approved
- Database backup and rollback path confirmed
- Known-good filesystem artifact or release package available for rollback

Commands:

```
# Pull code and deploy
cd /home/wordpress/public_html
git fetch origin
git checkout release/version-X.Y.Z
composer install --no-dev

# Apply schema updates only if the approved release requires them
wp core update-db

# Verify release state without mutating plugin/theme versions at deploy time
wp plugin list --format=table --fields=name,status,version
wp theme list --format=table --fields=name,status,version
wp core is-installed && echo "OK"
curl -I https://[CUSTOMIZE: example.com]
```

> **WARNING:** Do not run `wp plugin update --all` or `wp theme update --all` during deployment. Patch plugins and themes through the dedicated update workflow so rollback remains tied to known-good artifacts.

Expected Output:

- Target release is checked out.

- Composer dependencies install without fatal errors.
- Site health checks return successful status.

Rollback:

Use Section 8.3 if post-deploy verification fails. Rollback depends on a known-good release artifact or filesystem backup, not on live WordPress.org update state.

Verification:

- Homepage and the Dashboard login return expected HTTP status.
- Core workflows function (login, content read/write, critical integrations).

Escalate If:

- Deployment cannot complete within change window.
- Core update-db step fails.
- Service remains degraded after rollback.

8.2 Database Migration

Time Estimate: 30-60 minutes

> **WARNING:** Always backup before any database migration. This is high-risk.

Purpose: Migrate WordPress database safely between environments while preserving integrity and minimizing outage risk.

Prerequisites: - Database backup created and verified - Change window scheduled - Rollback plan documented - All users notified

Commands:

1. **Export Database from Source**

- ```
  # From old server
  wp db export /home/wordpress/backup/migration-source-$(date +%Y%m%d).sql --single-transaction

  # Or using mysqldump
  mysqldump -u [CUSTOMIZE: user] -p --single-transaction --quick \
    [CUSTOMIZE: database_name] > /home/wordpress/backup/migration-source-$(date +%Y%m%d).sql
  ```

2. **Transfer Database**

- ```
  # Secure copy to new server
  scp /home/wordpress/backup/migration-source-*.sql [CUSTOMIZE: user@new-server]:/tmp/
  ```

3. **Prepare New Database**

- ```
  # On new server - create empty database
  wp db create

  # Or create DB/user manually as root (single command)
  mysql -u root -p -e "CREATE DATABASE [CUSTOMIZE: new_database_name]; CREATE USE
  ```

4. **Import Database**

- ```
  # On new server
  wp db import /tmp/migration-source-*.sql

  # Or using mysql command
  mysql -u [CUSTOMIZE: user] -p [CUSTOMIZE: database] < /tmp/migration-
  source-*.sql
  ```

5. **Update WordPress URLs** (if hostname changed)

- ```
  # WARNING: This modifies all URL references across all database tables.
  wp search-replace "https://old.example.com" "https://new.example.com" -
  -all-tables

  # WARNING: Incorrect URLs here will make the site inaccessible via the Dashboard
  wp option update home "https://[CUSTOMIZE: new.example.com]"
  wp option update siteurl "https://[CUSTOMIZE: new.example.com]"
  ```

6. **Verify Database**

- ```
  # Check integrity
  wp db check

  # Verify tables count
  wp db tables --all-tables --format=table

  # Check for errors
  wp db query "SHOW ENGINE INNODB STATUS\G" | head -50
  ```

7. **Clear Caches**

- ```
  wp cache flush
  # Plugin-dependent — uncomment the cache plugin(s) in use:
  # wp w3-total-cache flush all
  # wp redis flush-db
  ```

Expected Output:
  ```

- Import completes without SQL errors.
- URL replacement is accurate for changed domains.
- `wp db check` passes and site is reachable.

Rollback:

```
# Restore source backup onto target if migration validation fails
wp db import /home/wordpress/backup/migration-source-YYYYMMDD.sql
```

Use Section 8.3 if deployment-level rollback is required.

Verification:

- Watch error logs for 15-30 minutes.
- Test critical pages and user login.
- Confirm media URLs and canonical links resolve correctly.

Escalate If:

- Import fails with unrecoverable SQL errors.
- Data mismatch is detected after migration.
- Post-migration errors persist after cache flush and URL correction.

8.3 Rollback Procedure

Time Estimate: 15-30 minutes

> **CRITICAL:** Test rollback procedure quarterly.

Scenario: Deployment introduced critical errors and site is down or broken.

Purpose: Restore service quickly after a failed deployment and preserve evidence for root-cause analysis.

Prerequisites:

- Known-good release/tag or backup artifact
- Pre-deployment database backup available
- Incident owner assigned

Commands:

1. **Identify Current State**

- ```
  wp core version
  git log --oneline -5
  git status
  ```
  ```

2. **Revert Code to Previous Release**

- ```
  # If using git and the release artifact itself is still intact
  git checkout previous-release-tag
  # OR
  git revert HEAD --no-edit  # Revert last commit

  # OR restore the known-good filesystem artifact
  tar -xzf /home/wordpress/backup/wordpress_TIMESTAMP.tar.gz -C /
  ```

3. **Revert Database if Needed**

- ```
  # Restore from pre-deployment backup
  gunzip < /home/wordpress/backup/pre-deploy-TIMESTAMP.sql.gz | wp db import -


  # Note: There is no transactional rollback for wp db import.
  # Rollback depends on a verified pre-deployment backup or filesystem artifact.
  ```

4. **Verify Rollback**

- ```
  wp core version
  curl -I https://[CUSTOMIZE: example.com]
  tail -30 wp-content/debug.log
  ```

5. **Investigate Root Cause**

- ```
  # Review error logs
  tail -100 /var/log/php-errors.log

  # Check component state
  wp plugin list --status=inactive --format=table
  wp theme list --status=active --format=table

  # Review recent changes
  git diff [old-version]...[new-version] | head -100
  ```

Expected Output:

- Site returns healthy HTTP status.
- Core functionality restored to pre-deployment behavior.
- Error rates return to baseline.

Rollback:

If first rollback attempt fails, proceed to full restore in Section 11.2.

Verification:

- Confirm homepage and the Dashboard login load.
- Validate critical business workflows.
- Confirm log noise/fatal errors drop to normal levels.

Escalate If:

- Rollback fails or extends beyond outage threshold.
- Data integrity cannot be confirmed.
- Repeated rollback attempts do not stabilize service.

Documentation Checklist:

- What went wrong
- When discovered
- Rollback time
- Root cause
- Prevention measures

Section 9: Content Operations

9.1 Media Management

Upload Folder Structure:

WordPress organizes uploads by year and month:

```
wp-content/uploads/
├── 2026/
│   ├── 01/
│   │   ├── image.jpg
│   │   ├── image-150x150.jpg  (thumbnail)
│   │   └── image-300x300.jpg  (medium)
│   └── 02/
│       └── ...
└── cache/  (plugin caching)
```

Image Optimization with WebP:

> **CAVEAT:** Not all browsers support WebP. Always maintain JPEG/PNG fallbacks.

```
# Install image optimization plugin
wp plugin install wp-smushit --activate
```

Or configure in Nginx for automatic WebP delivery:

```
location ~* ^.+\.(jpg|jpeg|gif|png)$ {
    # Serve WebP if browser accepts it and file exists
    if ($http_accept ~* "webp") {
        rewrite ^(.*)$ $1.webp break;
    }
    expires 30d;
    add_header Cache-Control "public, immutable";
}
```

Monitor Upload Folder Size:

```
# Check upload directory size
du -sh /home/wordpress/public_html/wp-content/uploads/

# Find large files
find /home/wordpress/public_html/wp-content/uploads/ -type f -
size +10M -exec ls -lh {} \;

# Clean up orphaned images (use caution)
wp media regenerate --yes  # Regenerate thumbnails
```

Offload Media to CDN (Optional):

```
# Install offload plugin
wp plugin install wordpress-unlimited-amazon-s3-media-library --
activate

# Configure S3 bucket
wp option update as3cf_settings '{
  "bucket": "[CUSTOMIZE: my-bucket]",
  "region": "[CUSTOMIZE: us-east-1]"
}'
```

9.2 Publishing Workflow

Content Calendar (Monthly Table Example):

Date	Post Title	Author	Status	Notes
Feb 1	Spring Campaign Launch	Jane	Draft	Needs final review
Feb 5	Product Update	Bob	Scheduled	Publishing at 9 AM
Feb 10	Blog Series Pt. 1	Alice	Published	2,500 views

Create and Schedule Posts:

```
# Create draft post
wp post create --post_type=post --post_title='New Post' \
  --post_content='Post content here' --post_status=draft

# Schedule post for future publishing
wp post create --post_type=post --post_title='Scheduled Post' \
  --post_content='Content' --post_status=future \
  --post_date='2026-02-20 09:00:00'

# List scheduled posts
wp post list --post_status=future --format=table
```

Review Before Publishing:

1. Content review for accuracy and tone
2. Check for broken links (use an external tool or broken-link-checker plugin)
3. Review SEO metadata (if using Yoast)
4. Verify images display correctly
5. Test on mobile device
6. Check URL/permalink is correct

Publish Post:

```
# Transition draft to published
wp post update [POST_ID] --post_status=publish

# Verify publication
wp post get [POST_ID] --format=table
```

9.3 Taxonomy and Permalink Management

Categories and Tags:

```
# List all categories
wp term list category --format=table

# Create new category
wp term create category "New Category" --description="Category description"

# Update category
wp term update category [TERM_ID] --name="Updated Name"

# Delete category (reassign posts)
wp term delete category [TERM_ID]  # WordPress reassigns posts to the default categor
```

Permalink Structure:

Current structure: [CUSTOMIZE: /%year%/%monthnum%/%postname%/]

```
# View current permalink structure
wp option get permalink_structure

# Change permalink structure (takes effect immediately)
wp rewrite structure '/%year%/%monthnum%/%postname%/'

# WARNING: This regenerates rewrite rules and may modify .htaccess or server config.
wp rewrite flush

# For 301 redirects from old URLs, configure at the web server level
# or use a redirect plugin's Dashboard screen (e.g., Redirection plugin)
```

> **WARNING:** Changing permalink structure requires 301 redirects from old URLs. Update internal links and notify users.

Custom Post Types:

```
# List custom post types
wp post-type list --format=table

# Create custom post type programmatically
wp plugin install [custom-plugin] --activate
```

```
# Register in functions.php
add_action('init', function() {
    register_post_type('portfolio', array(
        'public' => true,
        'label' => 'Portfolio',
        'supports' => array('title', 'editor', 'thumbnail'),
    ));
});
```

9.4 Search and Indexing

WordPress Search Optimization:

```
# Verify WP search is working
wp post list --s="keyword" --post_type=post --format=table

# Rebuild search index (plugin-dependent, e.g., ElasticPress or SearchWP)
# wp elasticpress index --setup

# Check for posts with no content (unfixed)
wp db query "SELECT ID, post_title FROM $(wp db prefix)posts WHERE post_content = ''
```

Search Engine Indexing:

```
# Check if site is visible to search engines
wp option get blog_public

# Enable search engine visibility
wp option update blog_public 1

# Verify core sitemaps are accessible (WordPress 5.5+)
curl -so /dev/null -w "%{http_code}" https://[CUSTOMIZE: example.com]/wp-
sitemap.xml
```

WP-CLI Search and Replace:

```
# Search for content
wp search-replace "old-text" "new-text" --dry-run  # Preview changes

# WARNING: This permanently modifies matching content across all tables.
```

```
wp search-replace "old-text" "new-text"

# WARNING: This permanently modifies matching content in the posts table.
wp search-replace "old" "new" "$(wp db prefix)posts"
```

Section 10: Incident Response

Lifecycle metadata for incident response procedures is tracked in Appendix E.

Prerequisites for all incident procedures below: - SSH access to the production host with sudo privileges - WP-CLI 2.5+ installed and accessible in `$PATH` (verify: `wp cli version`) - Access to monitoring dashboards and log aggregation - Contact list for on-call personnel (see Section 10.4) - Verified recent backup exists and restore path is documented (see Section 11.2)

10.1 Severity Classification

Severity	Impact	Response Time	Example	Relevant Playbook
Critical (P1)	Site completely down or major data loss	15 minutes	Database corruption, ransomware, all users locked out	§10.2, §10.3
High (P2)	Core functionality broken, partial outage	30 minutes	Plugin causing 500 errors, login broken, media not loading	§10.2, §10.5
Medium (P3)	Non-critical feature broken or degraded	4 hours	Search not working, theme display issues	§10.5

Severity	Impact	Response Time	Example	Relevant Playbook
Low (P4)	Minor issues, cosmetic or UX	1-2 business days	Typo, formatting issue, slow report generation	—

10.2 Site Down / 500 Error Triage

Time Estimate: 5-15 minutes to identify root cause (longer if plugin isolation or database repair is required)

Use this playbook when: uptime monitor fires a 5xx alert, or users report the site is unreachable.

Alert Meaning: `500/502/503` on the public site or `/wp-admin/` indicates the WordPress stack is degraded or unavailable (application, PHP-FPM, database, or host-level failure).

Customer Impact: Users may be unable to read content, authenticate, publish, or complete transactions. Treat as P1/P2 depending on scope and duration.

Diagnosis:

1. **Confirm outage and scope**

- ```
  curl -I https://[CUSTOMIZE: example.com]
  curl -I https://[CUSTOMIZE: example.com]/wp-admin/
  curl -s -w "%{http_code}\n" -o /dev/null https://[CUSTOMIZE: example.com]
  ```

2. **Check CDN/WAF layer (if applicable)**

- ```
  # If using a CDN or WAF, check for edge-layer errors:
  curl -sI https://[CUSTOMIZE: example.com] | grep -iE "cf-ray|server|x-cache"
  # Check CDN status page: [CUSTOMIZE: https://www.cloudflarestatus.com/]
  ```

3. **Check service health and host resources**

- ```
  systemctl status nginx
  systemctl status [CUSTOMIZE: php_fpm_service]
  systemctl status mysql
  top -bn1 | head -20
  free -h
  df -h | grep -E "^/dev|^Filesystem"
  ```
  ```

4. **Inspect recent errors**

- ```
  tail -30 /var/log/php-errors.log
  tail -30 /var/log/nginx/error.log
  tail -30 /var/log/mysql/error.log
  tail -30 [CUSTOMIZE: /home/wordpress/public_html]/wp-content/debug.log
  ```

5. **Test plugin isolation**

- ```
  wp plugin list --status=active --fields=name,status,version --
  format=table
  # WARNING: Deactivates ALL plugins including security, caching, and e-
  commerce.
  # On production with active transactions, consider testing a single suspect plug
  # Use only when service is already fully down.
  wp plugin deactivate --all
  curl -I https://[CUSTOMIZE: example.com]
  ```

6. **Validate database state**

- ```
  wp db check
  ```

7. **Check PHP memory constraints**

- ```
  php -r "echo ini_get('memory_limit').PHP_EOL;"
  ```

Immediate Mitigation:

1. If plugin isolation restores service, reactivate plugins one-by-one and keep the failing plugin disabled.
2. Restart core services in dependency order:

- ```
  sudo systemctl restart mysql
  sleep 5
  sudo systemctl restart [CUSTOMIZE: php_fpm_service]
  sleep 5
  sudo systemctl restart nginx
  ```

3. If memory pressure is confirmed, update `wp-config.php` and redeploy:

- ```
  define('WP_MEMORY_LIMIT', '256M');
  define('WP_MAX_MEMORY_LIMIT', '512M');
  ```

4. If `wp db check` reports errors, repair and optimize:

- ```
  wp db repair
  wp db optimize
  ```
  ```

Escalation:

- Escalate to Section 10.4 immediately if service remains down after mitigation attempts.
- Trigger backup restore path via Section 11.2 if recovery time exceeds outage threshold.
- If cloud/network signals indicate platform fault, escalate to hosting provider incident channel.

Recovery Validation:

```
curl -s -w "%{http_code}\n" -o /dev/null https://[CUSTOMIZE: example.com]
curl -s -w "%{http_code}\n" -o /dev/null https://[CUSTOMIZE: example.com]/wp-admin/
tail -20 /var/log/nginx/error.log
tail -20 /var/log/php-errors.log
```

Confirm Dashboard login and at least one critical business workflow (e.g., [CUSTOMIZE: test checkout, form submission, or user registration]) before closing incident.

10.3 Security Breach Response

Time Estimate: 1-4 hours depending on scope

Use this playbook when: security scanner detects malware, unexpected admin accounts appear, or the site redirects to an unknown domain.

> **CRITICAL:** If breach is suspected, act immediately. Data loss and reputation damage increase with every minute of delay.

This procedure implements the NIST SP 800-61r3 incident handling lifecycle described in the WordPress Security Hardening Guide §12.3.

Alert Meaning: Evidence suggests active compromise or unauthorized access (malicious files, account misuse, redirect behavior, or scanner-confirmed malware).

Customer Impact: Confidentiality, integrity, and availability are all at risk. This can require service isolation, user-facing communications, and credential revocation.

First action – declare the incident:

```
Severity: CRITICAL
Issue: [Description of attack]
Time Detected: [Time]
Affected User Data: [If known]
Escalation: [Your contact details]
```

Diagnosis:

1. **Contain exposure**

- ```
  # Take site offline to prevent further data exfiltration
  # Option 1: Redirect to maintenance page (requires WP-CLI 2.5+)
  wp maintenance-mode activate

  # Option 2: Block all traffic except admins
  # Add to .htaccess or nginx config:
  # deny all;
  # allow [CUSTOMIZE: your-ip];
  ```

2. **Determine scope of compromise**

- ```
  wp user list --format=table
  wp user list --role=administrator --format=table
  find [CUSTOMIZE: /home/wordpress] -name "*.php" -type f -mtime -
  7 -ls
  find [CUSTOMIZE: /home/wordpress] -name "shell.php" -o -
  name "admin.php" -o -name "tmp*.php"
  # Check for unauthorized SSH keys (common persistence mechanism)
  cat ~/.ssh/authorized_keys
  find /home -name authorized_keys -exec ls -la {} \;
  ```

3. **Capture forensic artifacts before cleanup**

- ```
  mkdir -p /root/forensics
  # For large sites, consider excluding wp-content/uploads or using incremental ba
  tar -czf /root/forensics/breach-evidence-$(date +%Y%m%d-%H%M%S).tar.gz \
    [CUSTOMIZE: /home/wordpress/public_html]
  wp db export /root/forensics/breach-evidence-db-$(date +%Y%m%d-
  %H%M%S).sql
  ```

4. **Perform security scans**

- ```
  # Wordfence scans must be initiated through the Dashboard at Wordfence > Scan
  # For CLI-based malware scanning, use dedicated tools:
  clamscan -r [CUSTOMIZE: /home/wordpress/public_html]/
  aide --check > /tmp/aide-report.txt
  ```

Immediate Mitigation:

1. Disable compromised plugins/themes and remove confirmed malicious files after evidence capture.

- `# WARNING: Deactivating and deleting a plugin removes it permanently. Capture fo`
  `wp plugin deactivate infected-plugin-name`
  `wp plugin delete infected-plugin-name`

2. Delete rogue accounts and reassign their content.

- `# WARNING: This permanently deletes the user account. Content is reassigned, not`
  `wp user delete [SUSPICIOUS_USER_ID] --reassign=[ADMIN_ID]`

3. Reset privileged credentials and terminate active sessions.

- ```
  wp user list --role=administrator --field=user_login | while read user; do
    NEW_PASS=$(openssl rand -base64 16)
    wp user update "$user" --user_pass="$NEW_PASS"
    echo "Reset $user — communicate new password via secure channel"
  done

  # Invalidate all existing sessions (built-in WP-CLI command, no plugin required
  wp user list --field=ID | xargs -I {} wp user session destroy {} -
  -all

  # Review and revoke application passwords for affected privileged users
  wp user application-password list [USER_ID] --fields=uuid,name,created,last_us
  -format=table
  wp user application-password delete [USER_ID] [UUID]
  ```

4. Patch vulnerable components.

- ```
  wp core update
  wp plugin update [AFFECTED_PLUGIN_SLUG]
  wp theme update [AFFECTED_THEME_SLUG]
  ```

Escalation:

- Escalate immediately via Section 10.4 to Security Officer and Incident Commander.
- If regulated data may be exposed, involve legal/compliance workflow before public disclosure.
- If compromise cannot be contained quickly, execute disaster recovery path in Section 11.2.

Recovery Validation:

Run active monitoring for at least one full monitoring window (minimum 30 minutes):

```
# Watch for re-compromise indicators in real time
tail -f /var/log/nginx/access.log | grep -E -i "shell|admin|eval"
```

```
watch -n 5 'ps aux | grep php'

# Check for suspicious cron jobs added during breach
crontab -l
wp cron event list
```

Then confirm recovery:

```
wp user list --role=administrator --format=table
wp cron event list
tail -40 /var/log/nginx/access.log
tail -40 /var/log/php-errors.log
```

Then confirm:

- no unauthorized admin accounts remain;
- malware scans return clean results;
- site behavior is normal for at least one monitoring window;
- incident report is completed in Section 10.6.

After all validation checks pass, initiate a full post-incident review per Section 10.6, including affected-user notification and a security audit schedule. For affected-user communications, follow the editorial standards in the WordPress Security Style Guide §7 for vulnerability severity language and disclosure practices.

10.4 Incident Roles and Escalation Path

Incident Role Cards:

Role	Primary Responsibilities	Handoff / Escalation Trigger
Incident Commander	Declares severity, sets priorities, owns timeline, and approves incident closure.	Escalate to management when SLA or customer impact thresholds are exceeded.
Technical Lead	Runs diagnosis and mitigation steps, assigns technical tasks, confirms service recovery.	Escalate when root cause is unknown after initial triage window.
Communications Lead	Sends internal/customer updates, keeps status page and stakeholders informed.	Escalate when legal/compliance review is required for messaging.

Role	Primary Responsibilities	Handoff / Escalation Trigger
Scribe	Maintains incident timeline, captures decisions/actions, records follow-ups for postmortem.	Escalate when critical timeline details are missing or conflicting.

> **Small teams:** One person may fill multiple roles. At minimum, separate the Incident Commander (decision authority) from the Technical Lead (hands-on-keyboard). The Scribe role should be filled by whoever is least busy, but the IC is responsible for ensuring the timeline is captured.

Contact Escalation Order:

1. **On-Call SysAdmin** (respond within 30 min)
    - Phone: [CUSTOMIZE]
    - Email: [CUSTOMIZE]
    - Slack: [CUSTOMIZE]
2. **On-Call Database Admin** (respond within 60 min)
    - Phone: [CUSTOMIZE]
    - Email: [CUSTOMIZE]
    - Slack: [CUSTOMIZE]
3. **Security Officer** (for security incidents)
    - Phone: [CUSTOMIZE]
    - Email: [CUSTOMIZE]
    - Slack: [CUSTOMIZE]
4. **Manager/Director** (for critical incidents)
    - Phone: [CUSTOMIZE]
    - Email: [CUSTOMIZE]
    - Slack: [CUSTOMIZE]
5. **Incident Commander** (if declared critical)
    - Phone: [CUSTOMIZE]
    - Email: [CUSTOMIZE]
    - Slack: [CUSTOMIZE]

> **Note:** The ordering above reflects escalation depth, not engagement sequence. The Incident Commander role is typically assumed by the On-Call SysAdmin at incident start and formally handed off when a senior IC is engaged.

10.5 Performance Degradation

Time Estimate: 15-30 minutes to identify cause

Use this playbook when: response time exceeds SLO thresholds, resource utilization alerts fire, or users report the site is slow.

Alert Meaning: Latency, resource saturation, or query contention indicates degraded but not fully unavailable service. Common triggers: CPU sustained above 80%, memory above 90%, page load exceeding 2 seconds, or slow-query volume spikes.

Customer Impact: Users experience slow page loads, failed submissions, and reduced admin productivity; prolonged degradation can cascade into full outage.

Diagnosis:

1. **Measure current system and application performance**

- ```
  uptime
  top -bn1 | head -15
  free -h
  time curl -s https://[CUSTOMIZE: example.com] > /dev/null
  ```

2. **Identify primary bottleneck**

- ```
  tail -50 /var/log/mysql/slow.log
  # Group recent requests by HTTP status code (field $9 in combined log format)
  tail -1000 /var/log/nginx/access.log | awk '{print $9}' | sort | uniq -
  c | sort -rn | head
  # If using a custom log format with $request_time as last field, also check:
  # tail -1000 /var/log/nginx/access.log | awk '{print $NF}' | sort -
  n | tail -20
  ```

3. **Check common WordPress-specific causes**

- ```
  df -h /home/wordpress
  wp db query "SELECT ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS '
  wp post list --orderby=post_date --order=DESC --posts_per_page=100 -
  -format=table
  wp cron event list
  ps aux | grep "wp cron"
  # Check autoloaded options size (values over 1MB indicate bloat)
  ```
  ```

```
wp db query "SELECT SUM(LENGTH(option_value)) AS autoload_bytes FROM $(wp db pr
wp db query "SELECT option_name, LENGTH(option_value) AS size FROM $(wp db prefi
```

If WP-Cron is a recurring bottleneck, see the WordPress Security Hardening Guide §7.2 for guidance on replacing it with a system cron job, and Section 6.6 for the operational procedure.

Immediate Mitigation:

1. Reduce immediate load and clear caches.

- ```
  wp cache flush
  # Plugin-dependent — uncomment the cache plugin(s) in use:
  # wp w3-total-cache flush all
  # wp redis flush-db
  ```

2. Run safe database hygiene steps.

- ```
  wp db optimize
  wp transient delete --expired
  ```

3. If a plugin is identified as the bottleneck, disable it and validate service response.

Escalation:

- Escalate via Section 10.4 if response times remain above SLO after mitigation window.
- Escalate to DBA when slow query volume persists despite cache/optimization actions.
- Escalate to infrastructure owner when host CPU/memory/disk saturation cannot be relieved at application layer.

Recovery Validation:

```
time curl -s https://[CUSTOMIZE: example.com] > /dev/null
uptime
free -h
# Expected: [CUSTOMIZE: max-age=3600 or similar — document your baseline cache-
control header]
curl -I https://[CUSTOMIZE: example.com] | grep -i Cache-Control
```

Confirm response time and host utilization return to baseline for at least 15 minutes.

10.6 Post-Incident Review

Complete within 24 hours of incident resolution

Documentation:

```
# Incident Report - [DATE]

## Summary
[1-2 sentence overview]

## Severity
[Critical/High/Medium/Low]

## Timeline
- HH:MM - Issue first detected
- HH:MM - Root cause identified
- HH:MM - Incident resolved
- HH:MM - Post-incident review started

## Root Cause
[Detailed explanation of what happened]

## Impact
- Duration: [X minutes]
- Users affected: [number or percentage]
- Data loss: [Yes/No and details]
- Revenue impact: [estimate if applicable]

## Resolution
[Steps taken to resolve]

## Prevention
1. [Action to prevent recurrence]
2. [Implement monitoring]
3. [Update runbook]

## Action Items
- [ ] [Task] - Owner: [Name] - Due: [Date]
- [ ] [Task] - Owner: [Name] - Due: [Date]

## Participants
<!-- Use role titles from Section 10.4: Incident Commander, Technical Lead, Communic
->
- [Name] - [Role]
- [Name] - [Role]
```

```
## Approval
Reviewed by: [Name]
Date: [Date]
```

Section 11: Disaster Recovery

Lifecycle metadata for disaster recovery procedures is tracked in Appendix E.

11.1 Recovery Objectives

Metric	Target	Notes
RTO (Recovery Time Objective)	1 hour	Maximum acceptable downtime
RPO (Recovery Point Objective)	6 hours	Maximum acceptable data loss
MTTR (Mean Time To Recovery)	30 minutes	Average time to recover
Backup Frequency	Every 6 hours	Database, 4x daily
Backup Retention	90 days	Monthly snapshots kept 1 year
Recovery Testing	Quarterly	Test recovery from backups

> **NOTE:** These targets assume proper backups and infrastructure in place. Regular testing is critical.

11.2 Full Site Restore from Backup

Time Estimate: 1-1.5 hours

When to Use: - Complete server failure or data corruption - Ransomware or major security breach - Accidental data deletion - Need to roll back to specific point in time

Prerequisites: - Backup verified and accessible - New server ready (or current server isolated for recovery) - Database backup in hand - WordPress file backup in hand

- Change owner assigned - If using managed hosting or managed database services, provider restore workflow or support path identified

Steps:

1. **Prepare Clean Server**

- ```
  # If using new server, ensure LEMP stack is installed
  # See Section 3.1 for architecture overview

  # If reusing the current server, stop services first and move the old web
  # root aside instead of deleting raw database files.
  sudo systemctl stop nginx
  sudo systemctl stop [CUSTOMIZE: php_fpm_service]
  sudo systemctl stop mysql

  sudo mv /home/wordpress/public_html /home/wordpress/public_html.pre-
  restore.$(date +%Y%m%d-%H%M%S)
  sudo mkdir -p /home/wordpress/public_html
  ```

  > **WARNING:** Do not delete the raw MySQL datadir with `rm -rf`. Recreate the database through MySQL or use the managed service's restore controls instead.

2. **Restore WordPress Files**

- ```
  # Option 1: From local backup
  tar -xzf /home/wordpress/backup/wordpress_TIMESTAMP.tar.gz -C /

  # Option 2: From S3
  aws s3 cp s3://[CUSTOMIZE: backup-bucket]/[CUSTOMIZE: example.com]/wordpress_T
  | tar -xz -C /

  # Verify files exist
  ls -la /home/wordpress/public_html/wp-config.php
  ```

3. **Restore Database**

- ```
  # Create database if not exists
  mysql -u root -p <<EOF
  CREATE DATABASE IF NOT EXISTS [CUSTOMIZE: wordpress_db];
  CREATE USER '[CUSTOMIZE: wp_user]'@'localhost' IDENTIFIED BY '[password]';
  GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, INDEX, DROP ON [CUSTOMIZE:
  FLUSH PRIVILEGES;
  EOF
  ```
  ```

```
# Restore from backup
# Option 1: From local backup
gunzip < /home/wordpress/backup/db_TIMESTAMP.sql.gz | wp db import -


# Option 2: From S3
aws s3 cp s3://[CUSTOMIZE: backup-bucket]/[CUSTOMIZE: example.com]/db_TIMESTAM
| gunzip | wp db import -
```

4. **Restore Permissions**

- ```
  # Set correct ownership
  sudo chown -R [CUSTOMIZE: wp_user]:www-data /home/wordpress/public_html

  # Set correct permissions (see Appendix A)
  find /home/wordpress/public_html -type f -exec chmod 644 {} \;
  find /home/wordpress/public_html -type d -exec chmod 755 {} \;
  chmod 400 /home/wordpress/public_html/wp-config.php
  ```

5. **Restore Configuration Files**

- ```
  # Verify wp-config.php has correct database credentials
  grep "DB_NAME\|DB_USER\|DB_HOST" /home/wordpress/public_html/wp-
  config.php

  # Update wp-config.php if needed
  wp config set DB_NAME [CUSTOMIZE: wordpress_db]
  wp config set DB_USER [CUSTOMIZE: wp_user]
  ```

6. **Verify Restoration**

- ```
  # Check WordPress core
  wp core is-installed && echo "WordPress installed"
  wp core version

  # Check database
  wp db check
  wp db tables --all-tables | head

  # Check plugins
  wp plugin list --status=active --format=table

  # Check theme
  wp theme list --status=active
  ```
  ```

7. **Update URLs** (if hostname changed)

```
# WARNING: Incorrect URLs here will make the site inaccessible via the Dashboard
wp option update home "https://[CUSTOMIZE: example.com]"
wp option update siteurl "https://[CUSTOMIZE: example.com]"

# WARNING: This modifies all URL references across all database tables.
wp search-replace "https://old.example.com" "https://[CUSTOMIZE: example.com]"
-all-tables
```

8. **Restore Services**

```
# Clear caches
wp cache flush
# Plugin-dependent — uncomment the cache plugin(s) in use:
# wp w3-total-cache flush all
# wp redis flush-db

# Restart services
sudo systemctl restart mysql
sudo systemctl restart [CUSTOMIZE: php_fpm_service]
sudo systemctl restart nginx

# Test site
curl -I https://[CUSTOMIZE: example.com]
```

9. **Post-Restore**

    - Reset privileged account passwords
    - Review and revoke application passwords for affected privileged users with `wp user application-password list` and `wp user application-password delete`
    - Check all privileged users can log in
    - Verify media loads correctly
    - Test form submissions
    - Monitor error logs for 30 minutes

11.3 Database-Only Recovery

Time Estimate: 20-30 minutes

When to Use: - Data corruption in database only - Accidental database edits/deletions - Database gone, but files are fine - Want to recover to specific point in time

Steps:

1. **Identify Target Backup**

- ```
  # List available database backups
  ls -lh /home/wordpress/backup/db_*.sql.gz

  # Or from S3
  aws s3 ls s3://[CUSTOMIZE: backup-bucket]/[CUSTOMIZE: example.com]/ | grep "db_
  ```

2. **Backup Current Database** (in case needed for comparison)

- ```
  wp db export /tmp/current-db-$(date +%Y%m%d-%H%M%S).sql
  ```

3. **Remove Only WordPress Tables**

- ```
  # WARNING: This drops all tables matching the current WordPress table prefix.
  # Use only after confirming the target prefix and after the backup in step 2 comp
  wp db clean --yes

  # Shared-database alternative: review the prefix first and use a manual, prefix-
  scoped drop list.
  mysql -u root -p [CUSTOMIZE: wordpress_db] <<EOF
  DROP TABLE IF EXISTS [CUSTOMIZE: table_prefix]commentmeta;
  DROP TABLE IF EXISTS [CUSTOMIZE: table_prefix]comments;
  -- ... drop all tables
  EOF
  ```

  > **WARNING:** Do not use `wp db reset --yes` on shared databases. It drops and recreates the entire database, not just WordPress tables.

4. **Import Backup**

- ```
  # From local backup
  gunzip < /home/wordpress/backup/db_TIMESTAMP.sql.gz | wp db import -


  # Or if backup is already uncompressed
  wp db import /home/wordpress/backup/db_TIMESTAMP.sql
  ```

5. **Verify Restoration**

- ```
  # Check database integrity
  wp db check

  # Count posts (compare to expected)
  wp post list --post_type=any --format=count
  ```
  ```

```
# Verify users
wp user list --format=table
```

6. **Update and Clear Caches**

- ```
  # If recovered to different time, update site metadata
  wp option update last_update_check '0'

  # Clear all caches
  wp cache flush
  # Plugin-dependent — uncomment the cache plugin(s) in use:
  # wp w3-total-cache flush all
  # wp redis flush-db
  ```

7. **Monitor Site**

   - Check error logs
   - Test user login
   - Verify recent posts are present (or missing if intended)

11.4 Server Migration

Time Estimate: 2-4 hours

Scenario: Moving WordPress to a new server (new data center, cloud provider, hardware upgrade, etc.)

Prerequisites: - New server with LEMP stack installed and configured - DNS access for domain - SSH access to both servers - Current database backup - WordPress file backup

Steps:

1. **Prepare New Server**

- ```
  # Verify all services running
  systemctl status nginx
  systemctl status [CUSTOMIZE: php_fpm_service]
  systemctl status mysql
  systemctl status redis

  # Create WordPress directory
  sudo mkdir -p /home/wordpress/public_html
  sudo chown [CUSTOMIZE: wp_user]:www-data /home/wordpress/public_html
  ```

2. **Backup Current Site**

- ```
  # Full backup before migration
  /usr/local/bin/wordpress-backup.sh

  # Verify backup integrity
  ls -lh /home/wordpress/backup/*_$(date +%Y%m%d).*
  ```

3. **Restore to New Server**

- ```
  # Follow Section 11.2 steps to restore files and database
  ```

4. **Update DNS**

- ```
  # Update DNS A record to point to new server IP
  # Verify DNS propagation
  nslookup [CUSTOMIZE: example.com]
  dig [CUSTOMIZE: example.com] +short

  # Wait for TTL to expire (typically 1-24 hours)
  ```

5. **SSL Certificate**

- ```
  # If using Let's Encrypt, certificate follows domain
  # Verify certificate on new server
  openssl s_client -connect [CUSTOMIZE: example.com]:443 -tls1_2

  # If manual certificate, copy to new server
  scp /etc/letsencrypt/live/[CUSTOMIZE: example.com]/* \
    root@new-server:/etc/letsencrypt/live/[CUSTOMIZE: example.com]/
  ```

6. **Test on New Server**

- ```
  # Test via IP address (before DNS updates)
  curl -H "Host: [CUSTOMIZE: example.com]" -I https://[new-ip-address]

  # Once DNS updated, test normally
  curl -I https://[CUSTOMIZE: example.com]
  ```

7. **Decommission Old Server**

- ```
  # Do NOT immediately shut down old server
  # Monitor new server for 24-48 hours first

  # After verification period, backup old server data
  tar -czf /home/wordpress/backup/old-server-final-$(date +%Y%m%d).tar.gz /home
  ```

```
# Then safely shut down
sudo shutdown -h now
```

8. **Post-Migration Verification**
    - All users can access site
    - HTTPS certificate valid
    - All plugins active
    - Database working
    - Backups configured on new server
    - Monitoring configured on new server

Appendix A: File Permissions Reference

A.1 WordPress File Permissions Table

Proper file permissions are critical for security. WordPress files should not be world-writable.

Path	Type	Owner	Group	Mode	Purpose
`/home/wordpress/public_html/`	dir	[CUSTOMIZE: wp_user]	www-data	755	Web root - readable by web server, owned by site user
`/home/word`	file	[CUSTOMIZE: wp_user]	www-data	644	PHP files - not world-writable
`/home/wordpress/public_html/wp-config.php`	file	[CUSTOMIZE: wp_user]	[CUSTOMIZE: wp_user]	400 (600 tempo-rary)	Config - most restrictive practical mode
`/home/word content/`	dir	[CUSTOMIZE: wp_user]	www-data	755	Content directory
`/home/wordpress/public_html/wp-content/uploads/`	dir	[CUSTOMIZE: wp_user]	www-data	755/775	User uploads (write as needed)
`/home/word content/up`	file	[CUSTOMIZE: wp_user]	www-data	644/664	Uploaded files

Path	Type	Owner	Group	Mode	Purpose
/home/wordpress/public_html/wp-content/plugins/	dir	[CUSTOMIZE: wp_user]	www-data	755	Plugins directory
/home/wordpress/public_html/wp-content/themes/	dir	[CUSTOMIZE: wp_user]	www-data	755	Themes directory
/home/wordpress/public_html/.htaccess	file	[CUSTOMIZE: wp_user]	www-data	644	Rewrite rules
/home/wordpress/backups/	dir	[CUSTOMIZE: wp_user]	[CUSTOMIZE: wp_user]	700	Backups - owner only

A.2 Permission Reset Script

Use this script to fix permissions after issues or new installations:

```
#!/bin/bash
# File: /usr/local/bin/wordpress-fix-permissions.sh
# Purpose: Reset WordPress file permissions to secure defaults

set -e

WP_ROOT="[CUSTOMIZE: /home/wordpress/public_html]"
WP_OWNER="[CUSTOMIZE: wp_user]"
WP_GROUP="www-data"  # web server group

if [ ! -d "$WP_ROOT" ]; then
    echo "Error: WordPress directory not found at $WP_ROOT"
    exit 1
fi

echo "Fixing WordPress permissions..."

# Directories: 755 (rwxr-xr-x)
echo "Setting directory permissions to 755..."
find "$WP_ROOT" -type d -exec chmod 755 {} \;

# Files: 644 (rw-r--r--)
echo "Setting file permissions to 644..."
find "$WP_ROOT" -type f -exec chmod 644 {} \;
```

```
# wp-config.php: 400 (r--------) steady-state
echo "Setting wp-config.php to 400 (use 600 temporarily only during managed edits)..
chmod 400 "$WP_ROOT/wp-config.php"

# .htaccess: 644
echo "Setting .htaccess to 644..."
if [ -f "$WP_ROOT/.htaccess" ]; then
    chmod 644 "$WP_ROOT/.htaccess"
fi

# Ownership
echo "Setting ownership to $WP_OWNER:$WP_GROUP..."
chown -R "$WP_OWNER:$WP_GROUP" "$WP_ROOT"

# Backup directory: owner-only
echo "Setting backup directory permissions..."
if [ -d "$WP_ROOT/../backup" ]; then
    chmod 700 "$WP_ROOT/../backup"
    chmod 600 "$WP_ROOT/../backup"/*
fi

echo "□ Permission reset complete"
echo "Verify with: ls -la $WP_ROOT"
```

Make executable:

```
sudo chmod +x /usr/local/bin/wordpress-fix-permissions.sh
```

Run when needed:

```
sudo /usr/local/bin/wordpress-fix-permissions.sh
```

Appendix B: wp-config.php Key Settings

B.1 Database Configuration

```
// Database configuration
define('DB_NAME', '[CUSTOMIZE: wordpress_db]');
```

```
define('DB_USER', '[CUSTOMIZE: wp_user]');
define('DB_PASSWORD', '[CUSTOMIZE: SecurePassword123!]');
define('DB_HOST', '[CUSTOMIZE: localhost]');
define('DB_CHARSET', 'utf8mb4');
define('DB_COLLATE', '');

// Optional: Non-standard port
define('DB_HOST', '[CUSTOMIZE: localhost:3307]');

// Optional: Socket connection
define('DB_HOST', '[CUSTOMIZE: localhost:/var/run/mysqld/mysqld.sock]');
```

B.2 Security Keys and Salts

Critical: Generate unique values using https://api.wordpress.org/secret-key/1.1/salt/

```
// Do not change these values! Generate new unique keys:
// Visit: https://api.wordpress.org/secret-key/1.1/salt/

define('AUTH_KEY',         'put your unique phrase here');
define('SECURE_AUTH_KEY',  'put your unique phrase here');
define('LOGGED_IN_KEY',    'put your unique phrase here');
define('NONCE_KEY',        'put your unique phrase here');
define('AUTH_SALT',        'put your unique phrase here');
define('SECURE_AUTH_SALT', 'put your unique phrase here');
define('LOGGED_IN_SALT',   'put your unique phrase here');
define('NONCE_SALT',       'put your unique phrase here');
```

> **WARNING:** Changing these keys will log out all users. Do this during maintenance window.

B.3 Security Constants

> **WARNING:** `DISALLOW_FILE_EDIT` is the baseline recommendation. Use `DISALLOW_FILE_MODS` only when updates are handled outside the WordPress Dashboard.

```
// Baseline: disable built-in theme/plugin editor in wp-admin
define('DISALLOW_FILE_EDIT', true);

// Optional hardened profile: disable plugin/theme installs and updates in Dashboard
// define('DISALLOW_FILE_MODS', true);
```

```
// Security: Disable XML-RPC — block at web server level (see Section 5.4)
// or use a must-use plugin: add_filter('xmlrpc_enabled', '__return_false');

// Security: Force HTTPS
define('FORCE_SSL_ADMIN', true);

// Security: REST API policy should be route-scoped, not globally blocked
// See Section 5.4 for endpoint-specific examples.
```

B.4 Debugging Constants

```
// Enable WordPress debugging (STAGING/DEV only, DISABLE in production)
define('WP_DEBUG', false);  // Set to true only for development
define('WP_DEBUG_LOG', false); // Set both WP_DEBUG and WP_DEBUG_LOG to true only d
define('WP_DEBUG_DISPLAY', false);  // Don't display errors on frontend

// Script debugging
define('SCRIPT_DEBUG', false);  // Load unminified JS/CSS (dev only)

// Database debugging
define('SAVEQUERIES', false);  // Track database queries (dev only)
```

B.5 Performance Constants

```
// Caching
define('WP_CACHE', true);  // Enable object caching

// Memory limits
define('WP_MEMORY_LIMIT', '256M');  // Per-request memory
define('WP_MAX_MEMORY_LIMIT', '512M');  // For background tasks

// Post revisions
define('WP_POST_REVISIONS', 5);  // Keep only 5 revisions per post

// Autosave interval (seconds)
define('AUTOSAVE_INTERVAL', 300);  // 5 minutes

// Trash retention (days)
define('EMPTY_TRASH_DAYS', 30);  // Auto-delete trash after 30 days
```

```
// Core updates
define('WP_AUTO_UPDATE_CORE', 'minor');   // Retain minor core auto-
updates (baseline behavior since WordPress 3.7)
```

To change JPEG compression quality, use the `jpeg_quality` filter in a must-use plugin:

```
add_filter('jpeg_quality', static function (): int {
    return 82;
});
```

B.6 WordPress Cron Constants

```
// Disable WordPress cron (use system cron instead)
define('DISABLE_WP_CRON', true);

// System cron replacement (add to crontab):
// */5 * * * * cd /home/wordpress/public_html && wp cron event run --
due-now > /dev/null 2>&1
// Block wp-cron.php at the web server level — see Section 6.6
```

B.7 Multisite Configuration (if applicable)

```
// Enable multisite (single blog)
define('WP_ALLOW_MULTISITE', true);

// Or if already multisite:
define('MULTISITE', true);
define('SUBDOMAIN_INSTALL', false);  // Use subfolders, not subdomains
define('DOMAIN_CURRENT_SITE', '[CUSTOMIZE: example.com]');
define('PATH_CURRENT_SITE', '/');
define('SITE_ID_CURRENT_SITE', 1);
define('BLOG_ID_CURRENT_SITE', 1);
```

B.8 Path Configuration (Optional)

```
// Override default paths (useful for non-standard installations)
define('WP_CONTENT_DIR', dirname(__FILE__) . '/wp-content');
define('WP_CONTENT_URL',     'https://[CUSTOMIZE:     example.com]/wp-
content');
```

```
define('WP_PLUGIN_DIR', WP_CONTENT_DIR . '/plugins');
define('WP_PLUGIN_URL', WP_CONTENT_URL . '/plugins');
```

B.9 Complete Recommended wp-config.php Template

```
<?php
/**
 * WordPress Configuration File
 *
 * [CUSTOMIZE: Your site description]
 * Created: [DATE]
 */

// ============================================================
// DATABASE CONFIGURATION
// ============================================================
define('DB_NAME', '[CUSTOMIZE: wordpress_db]');
define('DB_USER', '[CUSTOMIZE: wp_user]');
define('DB_PASSWORD', '[CUSTOMIZE: SecurePassword123!]');
define('DB_HOST', 'localhost');
define('DB_CHARSET', 'utf8mb4');
define('DB_COLLATE', '');

// ============================================================
// SECURITY KEYS AND SALTS
// ============================================================
// Generate at: https://api.wordpress.org/secret-key/1.1/salt/
define('AUTH_KEY',         '[CUSTOMIZE: unique-key]');
define('SECURE_AUTH_KEY',  '[CUSTOMIZE: unique-key]');
define('LOGGED_IN_KEY',    '[CUSTOMIZE: unique-key]');
define('NONCE_KEY',        '[CUSTOMIZE: unique-key]');
define('AUTH_SALT',        '[CUSTOMIZE: unique-key]');
define('SECURE_AUTH_SALT', '[CUSTOMIZE: unique-key]');
define('LOGGED_IN_SALT',   '[CUSTOMIZE: unique-key]');
define('NONCE_SALT',       '[CUSTOMIZE: unique-key]');

// ============================================================
// DATABASE TABLE PREFIX
// ============================================================
$table_prefix  =  '[CUSTOMIZE:  wp_]';    //  Default  prefix;  non-
default prefixes are optional obscurity control
```

```
// ============================================================
// LANGUAGE AND LOCALE
// ============================================================
// WPLANG is deprecated since WordPress 4.0. Set language via Settings > General or:
// wp language core install en_US && wp language core activate en_US

// ============================================================
// WORDPRESS SECURITY
// ============================================================
define('DISALLOW_FILE_EDIT',  true);    //  Baseline:  disable  built-
in file editor
// define('DISALLOW_FILE_MODS', true);  // Optional hardened profile; requires exte
define('FORCE_SSL_ADMIN', true);     // Require HTTPS for admin
// XML-RPC: disable at web server level or via must-use plugin
// (see Section 5.4 and WordPress Security Benchmark §4.4)

// ============================================================
// PERFORMANCE & CACHING
// ============================================================
define('WP_CACHE', true);
define('WP_MEMORY_LIMIT', '256M');
define('WP_MAX_MEMORY_LIMIT', '512M');
define('WP_POST_REVISIONS', 5);
define('AUTOSAVE_INTERVAL', 300);
define('EMPTY_TRASH_DAYS', 30);

// ============================================================
// CRON & BACKGROUND TASKS
// ============================================================
define('DISABLE_WP_CRON', true);  // Use system cron

// ============================================================
// DEBUGGING (PRODUCTION = false)
// ============================================================
define('WP_DEBUG', false);
define('WP_DEBUG_LOG', false);  // Set both WP_DEBUG and WP_DEBUG_LOG to true when d
define('WP_DEBUG_DISPLAY', false);
define('SCRIPT_DEBUG', false);

// ============================================================
```

```
// WORDPRESS CORE UPDATES
// ============================================================
define('WP_AUTO_UPDATE_CORE', 'minor');  // Auto-update minor versions

// ============================================================
// WORDPRESS CORE PATHS
// ============================================================
if (!defined('ABSPATH')) {
    define('ABSPATH', dirname(__FILE__) . '/');
}

// ============================================================
// WORDPRESS CONFIGURATION LOADED - Load wp-settings.php
// ============================================================
require_once(ABSPATH . 'wp-settings.php');
```

Appendix C: Change Log

Release History

This section documents all updates to this runbook and corresponding infrastructure changes.

Version 2.0 - February 2026 **Major Updates:** - Added comprehensive security hardening section (Section 5) - Expanded incident response procedures (Section 10) - Added disaster recovery procedures with RTO/RPO targets (Section 11) - Added automated monitoring and alerting section (Section 6.9) - Complete file permissions reference (Appendix A) - Detailed wp-config.php guidance (Appendix B)

Breaking Changes: - Set `DISALLOW_FILE_EDIT = true` as baseline; `DISALLOW_FILE_MODS` moved to optional hardened profile - Changed default post revision limit to 5 (from unlimited) - Updated PHP minimum version requirement to 8.3+

Infrastructure Changes: - Upgraded to MySQL 8.0 / MariaDB 10.6 - Added Redis caching layer - Implemented file integrity monitoring (AIDE) - Updated Nginx to 1.24+

Testing: - [CUSTOMIZE: Date] - Full disaster recovery test successful - [CUSTOMIZE: Date] - Security audit completed - [CUSTOMIZE: Date] - Performance testing done

Version 1.5 - August 2025 **Updates:** - Added database migration procedures (Section 8.2) - Expanded backup strategy and verification - Added WP-CLI command reference throughout - Updated SSL/TLS configuration for modern standards

Infrastructure: - SSL certificate renewal automated with Certbot - Added database backup redundancy (S3 + NAS) - Upgraded PHP to 8.0

Version 1.0 - February 2025 **Initial Release:** - Basic WordPress operations procedures - Deployment workflow - Backup and restore basics - Basic monitoring

---

Appendix D: Deprecated and Invalid Constants Guardrail

Do not add these symbols to `wp-config.php` hardening templates:

- `FORCE_SSL_LOGIN` (deprecated)
- `DISALLOW_PLUGIN_EDITING` (not a core constant)
- `DISALLOW_PLUGIN_ACTIVATION` (not a core constant)
- `SECURE_LOGGED_IN_COOKIE` (not a core constant)
- `define('XMLRPC_REQUEST', false)` (`XMLRPC_REQUEST` is request-context only)
- `WPLANG` (deprecated since WordPress 4.0; set language via Settings > General or WP-CLI `wp language core activate`)

Use `DISALLOW_FILE_EDIT` as baseline, and use `DISALLOW_FILE_MODS` only when deployment/update workflows are externalized.

Appendix E: Procedure Ownership and Validation Matrix

Use this matrix to track operational ownership and freshness for critical procedures.

Procedure	Owner	Last Tested	Review Cadence	Last Drill Date
6.2 WordPress Core Updates	[CUSTOMIZE: Role/Name]	[CUSTOMIZE: YYYY-MM-DD]	Monthly	N/A

Procedure	Owner	Last Tested	Review Cadence	Last Drill Date
6.3 Plugin and Theme Updates	[CUSTOM Role/Nam	[CUSTOMIZE: YYYY-MM-DD]	Weekly	N/A
6.6 WordPress Cron Management	[CUSTOMIZE Role/Name	[CUSTOMIZE: YYYY-MM-DD]	Monthly	N/A
8.1 Code Deployment	[CUSTOM Role/Nam	[CUSTOMIZE: YYYY-MM-DD]	Monthly	N/A
8.2 Database Migration	[CUSTOMIZE Role/Name	[CUSTOMIZE: YYYY-MM-DD]	Quarterly	[CUSTOMIZE: YYYY-MM-DD / N/A]
8.3 Rollback Procedure	[CUSTOM Role/Nam	[CUSTOMIZE: YYYY-MM-DD]	Quarterly	[CUSTOMIZE: YYYY-MM-DD]
10.2 Site Down / 500 Triage	[CUSTOMIZE Role/Name	[CUSTOMIZE: YYYY-MM-DD]	Quarterly	[CUSTOMIZE: YYYY-MM-DD]
10.3 Security Breach Response	[CUSTOM Role/Nam	[CUSTOMIZE: YYYY-MM-DD]	Quarterly	[CUSTOMIZE: YYYY-MM-DD]
10.5 Performance Degradation	[CUSTOMIZE Role/Name	[CUSTOMIZE: YYYY-MM-DD]	Quarterly	[CUSTOMIZE: YYYY-MM-DD]
11.2 Full Site Restore from Backup	[CUSTOM Role/Nam	[CUSTOMIZE: YYYY-MM-DD]	Quarterly	[CUSTOMIZE: YYYY-MM-DD]

Quick Reference Card (Printable)

For quick access during incidents, print this card and keep it near your desk.

```
╔═══════════════════════════════════════════════════════════════╗
║        WORDPRESS OPERATIONS QUICK REFERENCE                   ║
╚═══════════════════════════════════════════════════════════════╝

EMERGENCY CONTACTS:
- SysAdmin On-Call: [CUSTOMIZE: PHONE]
- Database Admin: [CUSTOMIZE: PHONE]
- Security Officer: [CUSTOMIZE: PHONE]

CRITICAL COMMANDS:
Check if site is down:       curl -I https://example.com
View recent errors:          tail -30 /var/log/php-errors.log
Check server status:         top -bn1 | head
Disable all plugins:         wp plugin deactivate --all
Clear all caches:            wp cache flush
Restore from backup:      gunzip < /home/wordpress/backup/db_TIMESTAMP.sql.gz | wp


IMPORTANT PATHS:
WordPress root:              /home/wordpress/public_html/
Configuration:              /home/wordpress/public_html/wp-config.php
Backups:                     /home/wordpress/backup/
Logs:                        /var/log/nginx/, /var/log/php-errors.log
Database:                    MySQL [CUSTOMIZE: wordpress_db]

SERVICE COMMANDS:
Restart Nginx:               sudo systemctl restart nginx
Restart PHP-FPM:         sudo systemctl restart [CUSTOMIZE: php_fpm_service]
Restart MySQL:               sudo systemctl restart mysql
Check service status:        sudo systemctl status [service]

ESCALATION PATH:
1. [CUSTOMIZE: First responder name]
2. [CUSTOMIZE: Manager name]
3. [CUSTOMIZE: Director name]
```

Related Documents

Companion Security Documents: - **WordPress Security Benchmark** — Prescriptive configuration controls (Level 1/Level 2) for the full WordPress stack: web server, PHP, database, core, authentication, file system, logging, supply chain, AI, server access, and Multisite. This runbook's hardening procedures implement many of the Benchmark's recommendations. - **WordPress Security Architecture and Hardening Guide** — Enterprise-focused security architecture covering threat landscape, OWASP Top 10 mapping, server hardening, REST API security, authentication, supply chain, incident response, and AI security. - **WordPress Security Style Guide** — Principles, terminology, and formatting conventions for writing about WordPress security. - **WordPress Security White Paper** — The official upstream document describing WordPress core security architecture, maintained at WordPress.org. - **WordPress Advanced Administration: Security** — The official WordPress documentation for security hardening, brute-force protection, HTTPS, backups, monitoring, and multi-factor authentication.

External References: - WordPress Official Documentation - WordPress Security Handbook - WP-CLI Command Reference - Nginx Documentation - MySQL Documentation - OWASP Top 10

Internal References: - Network Infrastructure Documentation: [CUSTOMIZE: Link] - Security Policies: [CUSTOMIZE: Link] - Incident Management Process: [CUSTOMIZE: Link] - Change Management Process: [CUSTOMIZE: Link] - Disaster Recovery Plan: [CUSTOMIZE: Link]

Glossary

Term	Definition
2FA / MFA	Two-factor / multi-factor authentication for privileged accounts
Action-gated reauthentication	A fresh identity check before sensitive or destructive actions, also known as sudo mode or step-up authentication
Application password	A revocable WordPress credential for REST API or XML-RPC access that is separate from the main login password

Term	Definition
Dashboard	The WordPress administrative UI, distinct from the `/wp-admin/` path when the UI rather than the URL is what matters
LAMP/LEMP	Linux, Apache/Nginx, MySQL, PHP - server stack
Multisite	A WordPress network with shared codebase and site-level admins plus network-level Super Admins
WP-CLI	Command-line interface for WordPress management
wp-config.php	WordPress configuration file containing database credentials and settings
Transient	Temporary data stored in WordPress database or cache
cron	The operating-system scheduler used to run recurring host-level tasks
Plugin	Extension that adds functionality to WordPress
WP-Cron	WordPress's traffic-triggered task scheduler, distinct from system cron
Theme	Collection of templates and assets defining site appearance
RTO	Recovery Time Objective - maximum acceptable downtime
RPO	Recovery Point Objective - maximum acceptable data loss
MTTR	Mean Time To Recovery - average time to recover from outage
FIM	File Integrity Monitoring - detecting unauthorized file changes
CDN	Content Delivery Network - geographically distributed servers
WebP	Modern image format with better compression than JPEG/PNG
HTTP Status Codes	200 (OK), 301 (Moved), 400 (Bad Request), 403 (Forbidden), 404 (Not Found), 500 (Server Error), 503 (Service Unavailable)

Document Metadata

Document Title: WordPress Operations Runbook for [CUSTOMIZE: Domain/Instance/Environment/Au
Version: [CUSTOMIZE: Version Number] **Status:** [CUSTOMIZE: Draft/Active/Deprecated/Retired]
Last Updated: [CUSTOMIZE: Date] **Next Review Date:** [CUSTOMIZE: Date] **Document Owner:** [CUSTOMIZE: Operations Team]
Approval: [CUSTOMIZE: Manager Name and Date]

Change Request Process: 1. Submit your change request to [CUSTOMIZE: owner email]. 2. Include: section, change description, justification, testing done 3. The document will be updated, and the version will be incremented. 4. All stakeholders will be notified of changes. 5. Training will be scheduled if needed.

Feedback and Improvements: To suggest improvements to this runbook, please contact [CUSTOMIZE: owner email] with: - Identified section or procedure to improve - What was unclear or missing - Suggested improvements - Your contact information

---

END OF DOCUMENT

This WordPress Operations Runbook is a comprehensive guide for managing WordPress installations in production environments. Regular review and updates are essential. All procedures should be tested in staging before implementation in production.