VectraOps

Knowledge Library

CHECKLIST

ASP Checklist: Production Readiness for Classic ASP Applications

A practical checklist for reviewing classic ASP application readiness before production use. Verify server configuration, security controls, dependency behavior, logging, rollback safety, and operational ownership with clear evidence and acceptance criteria.

Programming - July 03, 2026

Purpose

Classic ASP applications often fail in production for avoidable reasons: weak IIS hardening, inconsistent COM registration, fragile database connectivity, missing error handling, and no rollback plan. This ASP checklist helps you verify whether an application is actually ready to go live, not just whether it works on a developer machine.

Use this checklist to decide whether the application is suitable for production use, what must be fixed before release, and what evidence should be captured during review. If you are comparing this with a broader .NET release review, a NET Checklist: Operational Readiness Review for Production Use can be useful for aligning readiness criteria across older and newer stacks.

How to use this checklist

Run the checklist in order, from hosting and runtime dependencies through security, observability, deployment, and recovery. For each phase, confirm the items, capture evidence, assign ownership, and record a pass/fail decision.

A review is only useful if each check is verifiable. Avoid subjective statements such as "looks fine" or "should be stable." Instead, collect configuration exports, screenshots, log samples, test results, and rollback notes.

1) Hosting and runtime foundation

Purpose

Confirm that the IIS/Windows hosting layer, script engine behavior, and required runtimes are configured consistently across environments. Many ASP incidents begin here, especially when a server was prepared manually or copied from an older machine.

Checklist items

? Confirm the application pool identity is explicitly set and has only the minimum file system, database, and network permissions required.

? Review the IIS site binding configuration and validate that the intended hostnames, certificates, and ports are active in the target environment.

? Validate that the application pool is configured with the expected 32-bit or 64-bit setting for the application’s COM components and ODBC/OLE DB dependencies.

? Confirm classic ASP is enabled only where required and is not exposed on unrelated sites or virtual directories.

? Review script timeout, request length, and upload limits to ensure they match the application’s actual operating requirements.

? Validate that the server time zone, locale, and code page settings match application assumptions for date, currency, and character encoding.

? Confirm that all required Windows features, IIS modules, and optional components are installed and documented.

Evidence to capture

- IIS site and application pool export
- Host binding and certificate details
- Runtime and feature inventory
- Screenshot or export of ASP and application pool settings
- Notes describing 32-bit or 64-bit dependency requirements

Acceptance criteria

- The application starts under the intended pool identity without manual intervention.
- Required classic ASP settings are enabled and isolated to the intended application scope.
- Version-sensitive behavior such as 32-bit COM access is confirmed, not assumed.
- No undocumented server-side setting is required for basic request handling.

Owner

Platform or Windows engineer

Review cadence

At initial deployment, after any IIS/OS change, and during quarterly production reviews

2) Application code and error handling

Purpose

Confirm that the code fails safely, surfaces actionable errors, and does not depend on hidden defaults. Classic ASP often behaves differently when error handling is incomplete or when code assumes a clean server state.

Checklist items

? Review the application for explicit On Error Resume Next usage and validate that every instance has a bounded scope and a checked error path.

? Confirm that page-level and component-level error handling produces meaningful diagnostics without exposing sensitive internals to users.

? Validate that input is validated server-side for required fields, length, allowed characters, and expected formats.

? Review any dynamic SQL construction and confirm that parameters are used where supported by the data access layer.

? Test that session state, cookie handling, and form submission logic remain stable after browser refresh, timeout, and back-button scenarios.

? Confirm that file upload, file download, and path handling logic rejects traversal sequences and unsafe extensions.

? Review any reusable include files to ensure they do not create hidden dependencies across pages.

Evidence to capture

- Code review notes for error handling and input validation
- Sample error log entries and user-facing error behavior
- Test results for common failure paths

- Static review notes for SQL construction and file path handling

Acceptance criteria

- Error handling is deterministic and does not hide failures from operators.
- User input is validated before use in queries, filesystem operations, or downstream components.
- No request path depends on an unreviewed include or a swallowed exception.
- Failure cases produce a useful log entry and a safe user response.

Owner

Application developer or code reviewer

Review cadence

At every release, after significant code changes, and after any security defect

3) Data access and external dependencies

Purpose

Verify that database connectivity, COM objects, file shares, SMTP relays, and downstream services behave predictably under production conditions. In classic ASP, a single missing provider or inaccessible share can break a critical user flow.

Checklist items

? Confirm the database connection string uses the correct server, database, authentication mode, and encryption requirements for the target environment.

? Validate that connection pooling, timeout values, and command timeout values are appropriate for real transaction lengths.

? Review all COM component registrations and confirm that the versions deployed on the server match the versions expected by the application.

? Test any file share, network path, or UNC dependency from the application pool identity, not only from an administrator account.

? Confirm that SMTP, message queue, and external API dependencies have documented endpoints, credentials, and failure behavior.

? Validate that external dependency failures are logged and do not leave partial records or duplicate submissions.

? Review all scheduled jobs or batch processes that the application depends on and confirm their run windows and failure notifications.

Evidence to capture

- Connection string and credential handling review
- Dependency inventory with owners and endpoints
- Test results from the application pool identity
- COM registration or component version notes
- Failure path logs for database and external service timeouts

Acceptance criteria

- The application connects successfully from the production host under the correct identity.
- Required COM and provider versions are present and documented.
- External dependency failures are detectable and recoverable.
- No hidden dependency exists that only works on a developer workstation.

Owner

Database administrator, infrastructure engineer, or application owner

Review cadence

At deployment, after dependency changes, and after any incident involving connectivity or component failure

4) Security and access control

Purpose

Confirm that the application limits exposure at the web server, application, and data layers. Classic ASP applications commonly inherit risk from broad IIS permissions, weak session handling, and overexposed administrative pages.

Checklist items

? Review authentication requirements and validate that anonymous, integrated, or form-based access is enabled only where intended.

? Confirm that administrative pages, maintenance scripts, and diagnostic endpoints are not publicly reachable.

? Validate that session cookies use the appropriate security attributes supported by the deployment context and browser compatibility requirements.

? Review authorization checks for privileged actions and confirm they are enforced server-side on every request path.

? Test that sensitive values such as passwords, connection strings, and tokens are not displayed in source, error pages, or logs.

? Confirm that upload and download permissions are restricted to the minimum required folders and file types.

? Validate that TLS is enforced on all authenticated or sensitive traffic and that insecure fallback paths are disabled or redirected safely.

? Review dependency on legacy authentication or directory permissions and document any environment-specific requirements.

Evidence to capture

- Authentication and authorization matrix
- Public endpoint inventory
- Security test results for sensitive pages and file access
- TLS and certificate configuration evidence
- Log samples showing rejected access attempts

Acceptance criteria

- Privileged actions require server-side authorization checks.
- Sensitive endpoints are not exposed without a justified need.
- No secrets are stored or echoed in unsafe locations.
- Transport security is enforced where required by policy or data sensitivity.

Owner

Security engineer or application owner

Review cadence

At release, after authentication changes, and during scheduled security reviews

5) Logging, monitoring, and observability

Purpose

Verify that operators can detect failures, diagnose user-impacting issues, and trace transactions without reading source code. If logs are missing or inconsistent, classic ASP incidents become manual investigations.

Checklist items

? Confirm that application events, errors, and key business actions are logged with timestamps, correlation data, and environment identifiers.

? Validate that log files or event sources are writable by the application identity and protected from unauthorized modification.

? Review whether log rotation, retention, and archival meet operational and compliance requirements.

? Test that a failed request produces a searchable log entry with enough context to reproduce the issue.

? Confirm that health checks or synthetic checks exist for the main user journeys and not only for the home page.

? Validate that alert thresholds are defined for repeated 5xx responses, authentication failures, database timeouts, and dependency outages.

? Review whether the application has a documented support path for collecting logs from production without disrupting service.

Evidence to capture

- Sample application logs and event records
- Monitoring rule definitions or screenshots
- Log retention and rotation configuration
- A failed-request trace with correlation data
- Support instructions for log collection

Acceptance criteria

- Operators can identify where a request failed and which dependency was involved.
- Logs are durable, attributable, and protected.
- At least one monitoring signal covers the core business workflow.
- Alerting exists for recurring failures that require human action.

Owner

Operations engineer or site reliability engineer

Review cadence

At deployment, after logging changes, and during incident post-review

6) Deployment, configuration, and rollback

Purpose

Confirm that the deployment process is repeatable, documented, and reversible. A working classic ASP application can still be unsafe to release if the deployment path depends on manual edits or undocumented server changes.

Checklist items

? Confirm that application files, IIS configuration, COM registrations, and database changes are deployed through a documented sequence.

? Review configuration files and environment-specific settings to ensure no production secret is hard-coded in source control or a shared folder.

? Validate that configuration drift between development, test, and production is identified before release.

? Test that the deployment can be repeated on a clean host or restored environment using the documented procedure.

? Confirm that the rollback plan includes code, configuration, database, and component changes where applicable.

? Validate that backup files, scripts, and package artifacts are versioned and retrievable for the exact release candidate.

? Review maintenance window expectations and confirm the business owner knows the service impact and approval path.

Evidence to capture

- Deployment runbook
- Versioned release package or artifact manifest
- Configuration delta report

- Backup and restore proof
- Rollback procedure and responsible owner

Acceptance criteria

- The deployment can be executed by a qualified operator without tribal knowledge.
- Rollback is feasible within the required recovery window.
- Environment-specific settings are documented and controlled.
- The release package matches the reviewed code and configuration baseline.

Owner

Release manager or deployment engineer

Review cadence

Every release and after any production rollback or failed deployment

7) Test coverage and release validation

Purpose

Verify that the specific production paths have been exercised in a controlled environment. For classic ASP, a few passing page loads are not enough if the application relies on stateful workflows, legacy components, or edge-case input.

Checklist items

? Confirm that test cases cover the main user journeys, error paths, and privileged actions.

? Validate that regression tests include session timeout, invalid input, unauthorized access, and dependency failure scenarios.

? Test the application under production-like permissions, not elevated administrator access.

? Review any manual verification script and confirm it includes database write, read, update, and rollback checks where relevant.

? Validate that browser compatibility requirements are known and documented if the application depends on legacy client behavior.

? Confirm that acceptance testing has been signed off by the business owner or service owner for the release scope.

? Cross-check the release test matrix against the production configuration to ensure the tested path is the deployed path.

Evidence to capture

- Test plan and executed test results
- UAT or sign-off record
- Permission context used during test execution
- Regression evidence for failure scenarios
- Release test matrix mapped to production settings

Acceptance criteria

- The critical path has been tested end-to-end in a production-like environment.
- Negative tests prove that validation and authorization fail safely.
- The deployed configuration matches the validated configuration.

- Release sign-off is explicit and recorded.

Owner

QA engineer, application owner, or release approver

Review cadence

Every release and after any change to permissions, dependencies, or client compatibility

Common mistakes to avoid

Classic ASP production reviews often miss the same problems. Watch for these patterns and treat them as release blockers until proven otherwise:

- Assuming a component works because it loads under an administrator account.
- Leaving On Error Resume Next in place without checking the error state.
- Treating a successful homepage load as proof that the whole site is healthy.
- Deploying a new release without testing file uploads, authentication, and database writes.
- Forgetting to validate the application pool identity against network shares and COM access.
- Hard-coding environment-specific configuration in source files.
- Skipping rollback planning because the release is "small."
- Ignoring log retention until an incident requires historical evidence.

Pass/fail decision rule

Use a simple rule so the review produces an operational decision, not just a notes document.

- Pass: All critical security, dependency, deployment, and rollback items are verified, and any remaining issues are low risk with documented workarounds and owners.
- Conditional pass: The application can be released only if named follow-up actions are completed before or immediately after deployment, with explicit approval from the owner and security or operations as needed.
- Fail: Any unresolved issue affects authentication, authorization, data integrity, dependency availability, rollback safety, or the ability to support the application in production.

A release should never move forward on the strength of informal assurances alone. If an issue cannot be verified, treat it as unresolved.

Readiness scoring

Use this lightweight scoring method to compare releases and identify weak areas.

- Score each phase from 0 to 2.
- 0 = not verified or not in place
- 1 = partially verified, with gaps or manual workarounds
- 2 = fully verified with evidence and owner sign-off
- Total possible score: 14

Score interpretation

- 12?14: Ready for production use, assuming no critical blockers remain.
- 9?11: Borderline; release only with documented conditional approval and tracked remediation.
- 0?8: Not ready; the application needs further work before production use.

Track the score over time. Repeated low scores in the same phase usually indicate a systemic process problem, not a one-off defect. If you are building a broader readiness process around application delivery, a Learning Implementation Roadmap Checklist can help structure evidence, ownership, and acceptance criteria, while a Learning Checklist for AI and Machine Learning Projects shows a similar evidence-driven approach in a different technical context.

Follow-up actions when the checklist is not green

When the checklist produces a fail or conditional pass, keep the remediation concrete and time-bound.

- Assign each open item to one owner with a due date.
- Document the exact server, site, component, or query that failed verification.
- Re-test only the affected phase first, then rerun the full checklist if the change touches shared dependencies.
- Update the deployment runbook if the issue revealed a missing operational step.
- Re-score the application after remediation so the readiness trend is visible.

Final verification reminder

A classic ASP application is production-ready only when the hosting layer, dependencies, security controls, deployment process, and rollback path have all been verified in the actual operating context. If any of those pieces are only "understood" but not proven, the application is not yet ready.

Use this guidance together with algorithms best practices and algorithm maturity to connect the workflow with related operational context already available on the site.

Related guides in this cluster

- Securing ASP.NET Core APIs with JWT Authentication and Claims Validation

Related guides in this cluster

- ASP.NET Core Rate Limiting and Throttling in APIs

Related guides in this cluster

- Implementing ASP.NET Core JWT Authentication with Refresh Tokens

Related guides in this cluster

- ASP.NET Core Secure API Authentication with JWT and Policy Authorization

Related guides in this cluster

- ASP.NET Core Authentication with JWT and Role-Based Access Control
- ASP.NET Core Identity Hardening with Secure Authentication Policies

Related guides in this cluster

- ASP.NET Core Authorization Policies with Role and Claim Checks

Related guides in this cluster

- ASP.NET Core Authentication with JWT Bearer Tokens and Claims-Based Authorization
- ASP.NET Core Rate Limiting for API Security and Throttling

Related guides in this cluster

- ASP.NET Core Authentication and Authorization Tutorial