nascenture®

SCALABLE WEB APPLICATION

Audit Checklist 2026

50+ Evaluation Points to Plan, Audit & Scale with Confidence

How to Use This Checklist

Work through each section and check off items your application currently meets. Use the Scoring Guide at the end to assess your overall scalability posture and prioritize your next steps.

01 Architecture

Modularity & Service Design

- ☐ Application is built on loosely coupled, independently deployable components *Microservices or modular monolith*
- ☐ Each service or module has a single, clearly defined responsibility
- ☐ Services communicate via versioned APIs — not shared databases or direct internal calls
- ☐ No single point of failure exists in the critical path of core user journeys
- ☐ The application is stateless at the application layer — session state stored externally *Redis, database*
- ☐ Data is owned by individual services — no shared monolithic database across service boundaries

Fault Tolerance & Resilience

- ☐ Circuit breakers implemented for all inter-service and third-party dependencies
- ☐ Bulkhead patterns isolate failures within resource pools
- ☐ Saga pattern (or equivalent) used for distributed transactions spanning multiple services
- ☐ Graceful degradation in place — core features remain available when non-critical services fail
- ☐ A service mesh or API gateway manages inter-service communication and observability
- ☐ Asynchronous messaging used for workloads that don't require real-time responses *Kafka, RabbitMQ, SQS*

API Design

- ☐ All APIs are versioned — breaking changes are never made to existing versions */v1/, /v2/*
- ☐ API contracts documented with OpenAPI / Swagger specifications
- ☐ Pagination implemented on all list endpoints — no unbounded data returns
- ☐ Idempotency implemented for all write operations — duplicate requests handled safely
- ☐ API deprecation policy documented — consumers given adequate notice of changes
- ☐ Rate limiting and throttling enforced at the API gateway level

02 Scaling & Load Management

Auto-Scaling

- ☐ Horizontal auto-scaling configured and tested — new instances provisioned automatically under load
- ☐ Auto-scaling policies scale down as aggressively as they scale up
- ☐ Load testing conducted at 2x, 5x, and 10x expected peak traffic
- ☐ Serverless functions used for burst, event-driven, or unpredictable workloads
- ☐ Traffic distributed across multiple availability zones within each region
- ☐ Multi-region deployment in place for applications serving global users

Load Balancing & Traffic Management

- ☐ Load balancers configured with health checks — unhealthy instances removed from rotation automatically
- ☐ Latency-based or geographic routing in place for global traffic distribution
- ☐ Connection draining configured — in-flight requests complete before instance removal
- ☐ DDoS protection configured at the network and application layer
- ☐ Web Application Firewall (WAF) in front of all public-facing endpoints

Caching Strategy

- ☐ In-memory caching layer in place for frequently read, rarely changing data *Redis, Memcached*
- ☐ CDN configured for all static assets — images, fonts, CSS, and JavaScript *Cloudflare, CloudFront*
- ☐ CDN caching extended to semi-dynamic and public API responses where appropriate
- ☐ Cache invalidation strategy documented and implemented — stale data handled deliberately
- ☐ Cache hit rate monitored and optimized — target above 80% for read-heavy applications
- ☐ Cache eviction policies defined — unbounded caches with no TTL identified and fixed

03 Database & Data Management

Query Performance

- ☐ Database indexes in place for all columns used in WHERE clauses, JOINs, and ORDER BY
- ☐ N+1 query problems identified and resolved through eager loading or query batching
- ☐ Slow query logging enabled — queries exceeding 100ms automatically captured and reviewed
- ☐ Query analysis run regularly *EXPLAIN ANALYZE in PostgreSQL, slow query log in MySQL*
- ☐ Read replicas configured to distribute read load away from the primary database
- ☐ Connection pooling implemented — no raw database connections opened per request *PgBouncer, ProxySQL*

Data Architecture

- ☐ Polyglot persistence approach used — the right database type for each workload
- ☐ Database sharding or partitioning strategy defined for tables projected to exceed 100M rows
- ☐ Large file and media storage uses object storage — not the application server filesystem *S3, GCS*
- ☐ Data lake or warehouse in place for analytical workloads — isolated from production database
- ☐ Event streaming in place for high-throughput data pipelines *Kafka, Kinesis*

Backup & Recovery

- ☐ Database backup and point-in-time recovery configured, tested, and monitored
- ☐ Recovery Time Objective (RTO) and Recovery Point Objective (RPO) defined and documented
- ☐ Disaster recovery runbook tested at minimum once per quarter
- ☐ Backup restoration tested — not just backup creation

04 Security & Compliance

Encryption & Access Control

- ☐ All data in transit encrypted with TLS 1.3 minimum
- ☐ All sensitive data at rest encrypted with AES-256 or equivalent
- ☐ Encryption keys managed separately from the data they protect *AWS KMS, HashiCorp Vault*
- ☐ Role-based access control (RBAC) implemented — users have least-privilege access
- ☐ Zero-trust architecture principles applied — no implicit trust based on network location
- ☐ Multi-factor authentication (MFA) enforced for all administrative and privileged access
- ☐ All secrets and credentials stored in a dedicated secrets manager — never in source control
- ☐ Mutual TLS (mTLS) enforced between internal services

Vulnerability Management

- ☐ Dependency vulnerability scanning runs automatically in the CI/CD pipeline *Snyk, Dependabot*
- ☐ Container images scanned for vulnerabilities before deployment *Trivy, Clair*
- ☐ Infrastructure security scanning in place for IaC configurations *Checkov, tfsec*
- ☐ Static application security testing (SAST) runs on every commit
- ☐ Penetration testing conducted at minimum annually by a qualified third party
- ☐ Runtime security monitoring in place for Kubernetes workloads *Falco*
- ☐ Incident response plan documented, tested, and accessible to the on-call team

Regulatory Compliance

- ☐ GDPR controls implemented — consent management, right to erasure, data portability *If applicable*
- ☐ HIPAA controls implemented — PHI encryption, audit logs, BAAs with vendors *If applicable*
- ☐ PCI DSS controls implemented — tokenization, network segmentation, quarterly scans *If applicable*
- ☐ SOC 2 controls implemented — security, availability, confidentiality framework *If applicable*
- ☐ Audit logs capture all user actions with timestamp, IP address, and session context
- ☐ Data residency requirements mapped to infrastructure — data stays in required regions
- ☐ Compliance requirements reviewed and updated at minimum annually

05 Observability & CI/CD

Observability

- ☐ Structured logging implemented across all services — logs centralized and searchable
- ☐ Key metrics collected and dashboarded for every service *Request rate, error rate, latency, saturation*
- ☐ Distributed tracing implemented using OpenTelemetry — traces available for all requests
- ☐ Alerting thresholds defined for all critical metrics — alerts route to the right people
- ☐ Service Level Objectives (SLOs) defined for every user-facing service
- ☐ Error budgets tracked and reviewed weekly by engineering and product leadership
- ☐ On-call runbooks exist for every alert type — responders know exactly what to do
- ☐ Business metrics tracked alongside technical metrics *Conversion rate, revenue per minute*
- ☐ Anomaly detection configured to surface unusual patterns before they escalate
- ☐ Post-incident reviews conducted within 48 hours of every significant incident

CI/CD Pipeline

- ☐ All infrastructure defined as code — no manual configuration through cloud consoles *Terraform, Pulumi*
- ☐ CI pipeline runs on every pull request — linting, unit tests, integration tests, security scanning
- ☐ CD pipeline deploys automatically to staging on merge — approval required for production
- ☐ Blue-green or canary deployment strategy in place — rollback takes under 5 minutes
- ☐ Feature flags used to decouple code deployment from feature release
- ☐ Database migrations are backward compatible — old and new app versions run simultaneously
- ☐ Performance regression tests run automatically on every deployment
- ☐ Deployment frequency measured — target is multiple deployments per week minimum
- ☐ Mean time to recovery (MTTR) measured and reviewed — target under 30 minutes
- ☐ Chaos engineering experiments run regularly in staging to identify failure modes proactively

06 Cost & Efficiency

Infrastructure Cost Management

- ☐ Cloud resource utilization reviewed monthly — over-provisioned instances right-sized
- ☐ Reserved or committed use discounts applied to predictable baseline workloads *30–60% savings*
- ☐ Spot or preemptible instances used for fault-tolerant batch and background workloads *70–90% savings*
- ☐ Data egress costs monitored and minimized through regional architecture and CDN usage
- ☐ Unused resources audited quarterly *Idle instances, orphaned storage, forgotten load balancers*
- ☐ Cost anomaly alerts configured to flag unexpected spending spikes immediately
- ☐ Infrastructure cost forecasts modeled for 2x and 5x growth scenarios

Unit Economics & Business Alignment

- ☐ Cost per active user per month tracked and trended
- ☐ Cost per API request or cost per transaction calculated and monitored
- ☐ Infrastructure cost as a percentage of revenue tracked quarterly
- ☐ Engineering incident hours per month tracked and trending downward
- ☐ Total Cost of Ownership (TCO) modeled for 3-year horizon — including engineering overhead
- ☐ Cost optimization findings reviewed in engineering leadership meetings monthly

Scoring Guide & Next Steps

Scoring Guide		
Score	**Assessment**	**Recommended Action**
85–100%	Scalability-Mature	Continuous improvement focus
70–84%	Scalability-Capable	Address gaps — prioritize security
50–69%	Scalability-Developing	Build a 6-month remediation roadmap
Below 50%	Scalability-At-Risk	Immediate action on highest-impact items

How to Calculate Your Score

Count the total number of items you checked off across all six sections, divide by 57 (total items), and multiply by 100 to get your percentage score.

Example: 40 items checked ÷ 57 total × 100 = 70% (Scalability-Capable)

Prioritizing Your Gaps

Not all unchecked items are equal. Prioritize gaps using this order:

1. Security & Compliance gaps — address immediately regardless of other priorities
2. Observability gaps — you can't fix what you can't see
3. Architecture gaps — highest long-term leverage but highest effort
4. Cost gaps — high ROI, usually achievable in days to weeks

Want to go deeper?

Read the full guide: Key Features Every Scalable Web Application Needs in 2026 — covering architecture, performance engineering, cost ROI, build vs. buy frameworks, and more.