BOOTS ON THE GROUND AI

AI Solutions for Small Business Owners

SENTRY PLAYBOOK

Catch Every Bug Before Your Users Do

Your complete guide to error monitoring with Sentry. From installation to production alerts, this playbook ensures every error is caught and debugged efficiently.

Built for teams that move fast.

BOOTS APPROVED

Version 1.0 | February 12, 2026

Boots On The Ground AI | bootsagentai.com

TABLE OF CONTENTS

Master Sentry error monitoring

WHY SENTRY?

Real-time error monitoring

What is Sentry?

Sentry is real-time error monitoring and performance tracking for your applications. Instead of waiting for users to report bugs, Sentry catches every error instantly, provides full stack traces, session replay, and performance metrics. It's like having a 24/7 quality assurance team that never sleeps.

Without Sentry vs With Sentry

Aspect	Without Sentry	With Sentry
Error Detection	Users report bugs via email/support	Instant alerts when errors occur
Debugging Info	Guess what went wrong	Full stack traces with code context
User Impact	You don't know who was affected	Session replay shows user experience
Performance Issues	Blind to slowdowns	Web Vitals dashboard and performance metrics
Root Cause	Manual investigation takes hours	Grouping shows related errors instantly
Alert Speed	Days to weeks	Seconds to your team
Release Tracking	No link between deploy and errors	Errors tied to exact releases/commits

BOOTS APPROVED

BOOTS SAYS: Small businesses lose money to bugs they never see. Sentry brings bugs into the light. Fix errors before they cost you customers. That's the Boots Standard.

10 GOLDEN RULES OF SENTRY

Principles for success

1 **Install Sentry before your first deployment**

Setup on day one. Not after you have customers. Prevention beats firefighting.

2 **Use the wizard CLI for setup**

npx @sentry/wizard -i nextjs handles configuration. Don't do it manually.

3 **Set up source maps for real code in errors**

Production errors show minified code unless you upload source maps. Do it.

4 **Configure alert rules - act on errors**

A silent error is a missed bug. Set up alerts so errors reach your team immediately.

5 **Use environments (dev, staging, production)**

Not all errors are equal. Separate environments so you focus on production issues.

6 **Tag releases to track which deploy caused which error**

npx @sentry/cli releases new VERSION links errors to exact code changes.

7 **Set sample rates wisely**

100% in dev, 10-20% in production. Capture enough to find issues without drowning in data.

8 **Never ignore Sentry alerts**

Mark issues as ignored only when genuinely fixed or false positives. Otherwise you miss regressions.

9 **Use Session Replay to see what users experienced**

Stack traces show what broke. Session replay shows why users saw it. Both matter.

10 **Review your Sentry dashboard weekly**

Make error monitoring a habit. Weekly reviews catch trends before they become crises.

CLI INSTALLATION & SETUP

Get Sentry running

Install with the Wizard

Recommended (Next.js):

```
npx @sentry/wizard -i nextjs
```

Manual Install:

```
npm install @sentry/nextjs
```

TIP

BOOTS EXPLAINS: The wizard CLI automates all setup. It creates config files, sets up error boundaries, and generates environment variables. Use it.

Essential Sentry CLI Commands

npx @sentry/wizard -i nextjs – Auto-configure Sentry for Next.js
npm install @sentry/nextjs – Manual Sentry package install
npx @sentry/cli login – Authenticate with your Sentry account
npx @sentry/cli releases new VERSION – Create a new release
npx @sentry/cli releases set-commits VERSION --auto – Link commits to release
npx @sentry/cli releases deploys VERSION new -e production – Mark deployment
npx @sentry/cli sourcemaps upload ./out – Upload source maps for error clarity
npx @sentry/cli events list – View recent error events

Wizard-Generated Files

sentry.client.config.ts – Client-side error handling
sentry.server.config.ts – Server-side error handling
sentry.edge.config.ts – Edge function error handling
app/global-error.tsx – Error boundary component
.env.sentry-build-plugin – Build configuration
.sentryrc – CLI authentication token

VERIFY SETUP

BOOTS EXPLAINS: After setup, test immediately. Trigger an error in development and verify it appears in your Sentry dashboard. If it doesn't, check your config.

CONFIGURATION FILES

sentry.client.config.ts

Annotated sentry.client.config.ts

```
import * as Sentry from "@sentry/nextjs"; Sentry.init({ dsn:
process.env.NEXT_PUBLIC_SENTRY_DSN, // Environment tags errors by deployment stage
environment: process.env.NODE_ENV, // Release links errors to code changes release:
process.env.NEXT_PUBLIC_RELEASE, // 20% sampling in production (reduce noise)
tracesSampleRate: process.env.NODE_ENV === "production" ? 0.2 : 1.0, // Enable session
replay replaysSessionSampleRate: 0.1, replaysOnErrorSampleRate: 1.0, });
```

Key Configuration Options:

- **dsn:** Your Sentry project identifier
- **environment:** Tag errors by deployment stage
- **release:** Link errors to code versions
- **tracesSampleRate:** Performance monitoring percentage
- **replaysSessionSampleRate:** Record % of user sessions
- **replaysOnErrorSampleRate:** Always record session when error occurs

Environment Variables (.env.local)

```
NEXT_PUBLIC_SENTRY_DSN=https://xxxxx@xxxxx.ingest.sentry.io/123456
NEXT_PUBLIC_RELEASE=1.0.0 SENTRY_ORG=your-org SENTRY_PROJECT=your-project
SENTRY_AUTH_TOKEN=sntrys_xxxxx
```

ENVIRONMENTS & RELEASES

Track errors by context

Environment Setup

Development (localhost)

100% sampling. Capture everything. You want to see every error during development.

Staging

50% sampling. You have limited staging traffic. Balance noise vs coverage.

Production

10-20% sampling. High traffic means a small percentage catches most issues without overwhelming your dashboard.

Release Tagging Strategy

A release is a version of your code deployed to production. Tagging releases lets you answer: 'Which deploy caused this error?' Tag releases with semantic versioning (1.0.0, 1.1.0, etc.).

Deploy workflow:

```
npm version patch
npx @sentry/cli releases new 1.0.1
npx @sentry/cli releases set-commits 1.0.1 --auto
npm run build && npm run deploy
npx @sentry/cli releases deploys 1.0.1 new -e production
```

ALERT RULES & NOTIFICATIONS

Never miss an error

Alert Rules Workflow

1. **Create an alert rule**
   Project Settings → Alert Rules → Create Alert Rule
2. **Set error conditions**
   Error level: Error or higher, Environment: production
3. **Set frequency threshold**
   Alert when 5 errors in 5 minutes (adjust to your tolerance)
4. **Add notification channel**
   Slack, email, PagerDuty, Discord, etc.
5. **Test the alert**
   Trigger an error and verify you receive notification
6. **Create critical alerts**
   Errors affecting critical paths (auth, payments) get immediate alerts
7. **Monitor performance**
   Set alerts for slow transactions (>5 seconds)

ALERT STRATEGY

PRO TIP: Set up multiple alert rules with different thresholds. Critical errors (payment processing) get immediate alerts. Non-critical errors get daily digest. Match alert frequency to error impact.

SESSION REPLAY & PERFORMANCE

See what users experienced

Session Replay

Session replay records your user's interaction: clicks, scrolls, network requests, and console errors. When an error occurs, you can watch exactly what the user did leading up to it. It's like having a video of the bug happening.

Enable in sentry.client.config.ts:

```
replaysSessionSampleRate: 0.1, // Record 10% of all sessions

replaysOnErrorSampleRate: 1.0, // Always record when error occurs
```

Web Vitals & Performance

Metric	Full Name	Target	What It Means
LCP	Largest Contentful Paint	< 2.5s	How fast main content renders
FID	First Input Delay	< 100ms	How fast app responds to clicks
CLS	Cumulative Layout Shift	< 0.1	How much content shifts during load
TTFB	Time to First Byte	< 600ms	Server response time
FCP	First Contentful Paint	< 1.8s	How fast any content appears

BOOTS APPROVED

BOOTS SAYS: Performance is part of user experience. Monitor Web Vitals every sprint. Slow pages lose customers.

COMMON ERROR PATTERNS

How to fix them

Problem: Hydration Mismatch

Solution: Server-rendered HTML doesn't match client. Solution: Use useEffect for client-only code.

Problem: ChunkLoadError

Solution: Dynamic import failed (usually network issue). Solution: Set up error boundary, retry logic.

Problem: TypeError: Cannot read property of undefined

Solution: Accessing property on null/undefined. Solution: Add null checks.

Problem: Network Error

Solution: Fetch/API request failed. Solution: Add retry logic, timeout handling, better error messages.

Problem: Unhandled Promise Rejection

Solution: Promise rejected without .catch(). Solution: Always chain .catch() or use try/catch.

Problem: Sentry not capturing errors

Solution: Config issue or error occurring before Sentry initializes. Solution: Check DSN, ensure Sentry.init() runs first.

DEBUG PRO TIP

PRO TIP: Create error handlers that capture context. Add breadcrumbs (Sentry.captureMessage) before critical operations so you see the path that led to the error.

PRODUCTION DEPLOYMENT CHECKLIST

Before you ship

Setup

- ☐ Sentry account created and project initialized
- ☐ @sentry/nextjs installed and configured
- ☐ DSN stored in environment variables (not hardcoded)
- ☐ sentry.client.config.ts and sentry.server.config.ts exist

Configuration

- ☐ Environment set to 'production' in sentry.init()
- ☐ Release version specified and incremented
- ☐ Sample rates configured (not 100%)
- ☐ Source maps generated and uploadable

Alerts

- ☐ Alert rules created for critical errors
- ☐ Slack/email notifications configured and tested
- ☐ Team members added to project
- ☐ On-call escalation configured if needed

Monitoring

- ☐ Dashboard created with critical metrics
- ☐ Performance monitoring enabled
- ☐ Session replay enabled with appropriate sampling
- ☐ First deployment error verified in Sentry

Security

- ☐ No test DSN in production (separate test project)
- ☐ Sensitive data filtered from Sentry (passwords, tokens, API keys)
- ☐ CORS whitelist configured to prevent abuse
- ☐ Sentry authentication token secured in CI/CD only

PRE-BUILD CHECKLIST

Before the next deploy

Error Handling

- ☐ Error boundaries wrap critical components
- ☐ Try/catch blocks around API calls
- ☐ Network errors have retry logic
- ☐ User-facing error messages are helpful (not technical jargon)

Performance

- ☐ Web Vitals checked and within targets
- ☐ No console errors or warnings in production
- ☐ Load times acceptable for all user types
- ☐ Database queries optimized (no N+1 problems)

Team

- ☐ All developers know how to access Sentry dashboard
- ☐ On-call engineer knows to check Sentry for issues
- ☐ Error review scheduled (weekly minimum)
- ☐ Sentry playbook shared with team

Documentation

- ☐ Critical user flows documented
- ☐ Known issues logged and tracked
- ☐ Runbook for common production errors created
- ☐ Release notes prepared with changes

FINAL CHECKPOINT

THE BOOTS STANDARD: You're ready to ship when every item above is checked. Monitor your errors. Fix them fast. Never ship blind.