Sentry Metrics Guide for CareLink AI

Overview

Sentry Metrics is now enabled for the CareLink AI project! This allows you to track application health, performance, and business metrics in real-time through the Sentry dashboard.

Dashboard URL: https://the-council-labs.sentry.io/explore/metrics/

Current Setup

- **Package**: `@sentry/nextjs` v10.38.0
- **Minimum Required Version**: 10.25.0 ✅
- **Metrics Status**: **ENABLED**
- **Configuration Files**:
- `sentry.client.config.ts` - Browser metrics
- `sentry.server.config.ts` - Server-side metrics
- `sentry.edge.config.ts` - Edge runtime metrics

Testing Metrics

Quick Test

Visit the test endpoint to send sample metrics:

```
curl https://your-app-url.com/api/test-sentry-metrics
```

Or open in browser:

```
https://your-app-url.com/api/test-sentry-metrics?testId=my-test-1
```

This will send:
- **Counters**: API call counts, test events
- **Gauges**: Active connections, memory usage
- **Distributions**: Response times, payload sizes, query times

Metric Types

1. Counters

Count discrete events (API calls, button clicks, orders)

Direct API:

```
import * as Sentry from '@sentry/nextjs';

Sentry.metrics.count('button.click', 1, {
  attributes: {
    button_id: 'submit-inquiry',
    page: '/homes',
  },
});
```

Using Utility:

```
import { trackBusinessEvent } from '@/lib/sentry-metrics';

trackBusinessEvent('inquiry.created', {
  home_id: 'home-123',
  plan: 'premium',
});
```

2. Gauges

Track current values that change over time (memory, connections, queue depth)

Direct API:

```
Sentry.metrics.gauge('queue.depth', 42, {
  attributes: {
    queue_name: 'email-notifications',
  },
});
```

Using Utility:

```
import { trackMemoryUsage } from '@/lib/sentry-metrics';

trackMemoryUsage(256, 'heap'); // 256 MB
```

3. Distributions

Track value distributions (response times, file sizes, latencies)

Direct API:

```
Sentry.metrics.distribution('api.response_time', 187.5, {
  unit: 'millisecond',
  attributes: {
    endpoint: '/api/homes',
    method: 'GET',
  },
});
```

Using Utility:

```
import { trackApiCall } from '@/lib/sentry-metrics';

const startTime = Date.now();
// ... perform API operation
const duration = Date.now() - startTime;

trackApiCall('/api/homes', 'GET', 200, duration);
```

Usage Examples

Track API Endpoints

```
// In your API route
import { trackApiCall } from '@/lib/sentry-metrics';

export async function GET(request: Request) {
  const startTime = Date.now();

  try {
    // Your logic here
    const data = await fetchData();

    const duration = Date.now() - startTime;
    trackApiCall('/api/homes', 'GET', 200, duration);

    return Response.json(data);
  } catch (error) {
    const duration = Date.now() - startTime;
    trackApiCall('/api/homes', 'GET', 500, duration);
    throw error;
  }
}
```

Track Database Queries

```
import { trackDatabaseQuery } from '@/lib/sentry-metrics';

const startTime = Date.now();
const homes = await prisma.assistedLivingHome.findMany({ /* ... */ });
const duration = Date.now() - startTime;

trackDatabaseQuery('SELECT', 'assistedLivingHome', duration);
```

Track User Authentication

```
import { trackAuthEvent } from '@/lib/sentry-metrics';

// On successful login
trackAuthEvent('login', user.role);

// On failed login
trackAuthEvent('failed_login');

// On registration
trackAuthEvent('register', 'FAMILY');
```

Track Business Events

```
import { trackBusinessEvent } from '@/lib/sentry-metrics';

// Track inquiries
trackBusinessEvent('inquiry.created', {
  home_id: homeId,
  source: 'website',
});

// Track tour bookings
trackBusinessEvent('tour.scheduled', {
  home_id: homeId,
  tour_type: 'in-person',
});

// Track payments
trackBusinessEvent('payment.completed', {
  amount: '1000',
  plan: 'premium',
});
```

Track Page Views

```
'use client';
import { useEffect } from 'react';
import { usePathname } from 'next/navigation';
import { trackPageView } from '@/lib/sentry-metrics';

export function PageViewTracker() {
  const pathname = usePathname();

  useEffect(() => {
    const startTime = performance.now();

    // Track when page is fully loaded
    const handleLoad = () => {
      const loadTime = performance.now() - startTime;
      trackPageView(pathname, loadTime);
    };

    if (document.readyState === 'complete') {
      handleLoad();
    } else {
      window.addEventListener('load', handleLoad);
      return () => window.removeEventListener('load', handleLoad);
    }
  }, [pathname]);

  return null;
}
```

Track Search Operations

```
import { trackSearch } from '@/lib/sentry-metrics';

const startTime = Date.now();
const results = await searchHomes(query);
const duration = Date.now() - startTime;

trackSearch('homes', results.length, duration);
```

Track File Uploads

```
import { trackFileOperation } from '@/lib/sentry-metrics';

const startTime = Date.now();
const file = await uploadToCloudinary(file);
const duration = Date.now() - startTime;
const sizeMB = file.size / (1024 * 1024);

trackFileOperation('upload', 'image', sizeMB, duration);
```

Track External API Calls

```
import { trackExternalApiCall } from '@/lib/sentry-metrics';

const startTime = Date.now();
try {
  await stripe.charges.create({ /* ... */ });
  const duration = Date.now() - startTime;
  trackExternalApiCall('stripe', 'create_charge', true, duration);
} catch (error) {
  const duration = Date.now() - startTime;
  trackExternalApiCall('stripe', 'create_charge', false, duration);
  throw error;
}
```

Track Cache Operations

```
import { trackCacheOperation } from '@/lib/sentry-metrics';

const cachedData = cache.get(key);
if (cachedData) {
  trackCacheOperation('hit', 'homes-list');
} else {
  trackCacheOperation('miss', 'homes-list');
  const data = await fetchData();
  cache.set(key, data);
  trackCacheOperation('set', 'homes-list');
}
```

Available Utility Functions

All functions are available in `/src/lib/sentry-metrics.ts`:

Function	Purpose	Example
`trackApiCall()`	Track API endpoint calls	`trackApiCall('/api/homes', 'GET', 200, 45)`
`trackDatabaseQuery()`	Track DB query performance	`trackDatabaseQuery('SELECT', 'homes', 23.5)`
`trackBusinessEvent()`	Track business metrics	`trackBusinessEvent('inquiry.created', {...})`
`trackAuthEvent()`	Track auth events	`trackAuthEvent('login', 'FAMILY')`
`trackPageView()`	Track page views	`trackPageView('/homes', 1250)`
`trackMemoryUsage()`	Track memory usage	`trackMemoryUsage(256, 'heap')`
`trackActiveConnections()`	Track connections	`trackActiveConnections(42, 'http')`
`trackSearch()`	Track search queries	`trackSearch('homes', 15, 87)`
`trackFileOperation()`	Track file ops	`trackFileOperation('upload', 'pdf', 2.5)`
`trackExternalApiCall()`	Track external APIs	`trackExternalApiCall('stripe', 'charge', true)`
`trackCacheOperation()`	Track cache hits/misses	`trackCacheOperation('hit', 'homes')`
`flushMetrics()`	Flush metrics immediately	`await flushMetrics()`

Viewing Metrics in Sentry

1. **Go to Sentry Dashboard**:
   - Navigate to: https://the-council-labs.sentry.io/explore/metrics/

2. **Explore Your Metrics**:
   - **Aggregates Tab**: View trends and totals across time
   - **Samples Tab**: See individual metric events with traces

3. **Filter and Group**:
   - Use attributes to filter (e.g., `endpoint=/api/homes`)
   - Group by attributes (e.g., `status_code`, `method`)

4. **Create Dashboards**:
   - Build custom dashboards with your metrics
   - Set up alerts for anomalies

Best Practices

1. Use Meaningful Names

```
// Good ✅
trackBusinessEvent('inquiry.submitted', {...});
trackBusinessEvent('payment.failed', {...});

// Avoid ❌
trackBusinessEvent('event1', {...});
trackBusinessEvent('thing', {...});
```

2. Add Relevant Attributes

```
// Good ✅
Sentry.metrics.count('api.error', 1, {
  attributes: {
    endpoint: '/api/homes',
    error_type: 'validation',
    status_code: '400',
  },
});

// Limited value ❌
Sentry.metrics.count('api.error', 1);
```

3. Keep Attribute Values Bounded

```
// Good ✅
trackApiCall('/api/homes', 'GET', 200); // bounded status codes

// Avoid ❌ - creates too many unique combinations
Sentry.metrics.count('user.action', 1, {
  attributes: {
    user_id: 'user-12345', // unbounded - too many unique values
  },
});
```

4. Use Appropriate Units

```
// Good ✅
Sentry.metrics.distribution('api.response_time', 187, {
  unit: 'millisecond',
});

Sentry.metrics.gauge('file.size', 2.5, {
  unit: 'megabyte',
});

// Less useful ❌ - no unit specified
Sentry.metrics.distribution('response_time', 187000); // microseconds? milliseconds?
```

5. Flush Before Exit

```
import { flushMetrics } from '@/lib/sentry-metrics';

// In cleanup or termination handlers
process.on('SIGTERM', async () => {
  await flushMetrics(2000);
  process.exit(0);
});
```

Common Metric Patterns

Performance Monitoring Pattern

```
async function monitoredOperation() {
  const startTime = Date.now();

  try {
    const result = await performOperation();
    const duration = Date.now() - startTime;

    // Track success
    Sentry.metrics.count('operation.success', 1);
    Sentry.metrics.distribution('operation.duration', duration, {
      unit: 'millisecond',
      attributes: { status: 'success' },
    });

    return result;
  } catch (error) {
    const duration = Date.now() - startTime;

    // Track failure
    Sentry.metrics.count('operation.failure', 1);
    Sentry.metrics.distribution('operation.duration', duration, {
      unit: 'millisecond',
      attributes: { status: 'error' },
    });

    throw error;
  }
}
```

Rate Limiting Pattern

```
function trackRateLimitStatus(exceeded: boolean, endpoint: string) {
  Sentry.metrics.count('rate_limit.check', 1, {
    attributes: {
      endpoint,
      exceeded: exceeded.toString(),
    },
  });
}
```

Health Check Pattern

```
function trackServiceHealth(service: string, healthy: boolean) {
  Sentry.metrics.gauge('service.health', healthy ? 1 : 0, {
    attributes: {
      service,
    },
  });
}
```

Troubleshooting

Metrics Not Appearing?

1. **Check Sentry Initialization**:
   `bash`
   `# Check if DSN is configured`
   `echo $SENTRY_DSN`
   `echo $NEXT_PUBLIC_SENTRY_DSN`

2. **Test Endpoint**:
   `bash`
   `curl https://your-app.com/api/test-sentry-metrics`

3. **Check Logs**:
   `bash`
   `# Look for Sentry initialization logs`
   `[Sentry Server] ✅ Initialized with features:`
   `[Sentry Server]   - Metrics: enabled`

4. **Verify Flushing**:
   ```typescript
   import { flushMetrics } from '@/lib/sentry-metrics';

// Send metrics
Sentry.metrics.count('test', 1);

// Force flush
await flushMetrics();
```

1. **Check Sentry Dashboard**:
   - Metrics may take a few seconds to appear

- Check the time range in the dashboard
- Verify you're looking at the correct project

Common Issues

Issue: "Sentry client not initialized"
- **Solution**: Verify `SENTRY_DSN` or `NEXT_PUBLIC_SENTRY_DSN` is set in environment variables

Issue: Metrics sent but not visible
- **Solution**: Check time range in dashboard, metrics may be buffered for up to 30 seconds

Issue: Too many unique metric combinations
- **Solution**: Reduce cardinality of attributes, avoid using unbounded values like user IDs

Resources

- **Sentry Metrics Documentation**: https://docs.sentry.io/platforms/javascript/guides/nextjs/metrics/
- **Sentry Dashboard**: https://the-council-labs.sentry.io/explore/metrics/
- **Test Endpoint**: `/api/test-sentry-metrics`
- **Utility Functions**: `/src/lib/sentry-metrics.ts`

Next Steps

1. ✅ Metrics are enabled
2. ✅ Test endpoint created
3. ✅ Utility functions available
4. 🎯 Add metrics to your API routes
5. 🎯 Track business events
6. 🎯 Monitor performance metrics
7. 🎯 Create custom dashboards in Sentry

Note: Metrics are automatically enabled in `@sentry/nextjs` v10.25.0+. Your current version (10.38.0) fully supports metrics without any package upgrades needed!