Unified Logging & Monitoring Standard (Revised)

Prototype:

1. Purpose & Scope

This standard defines unified observability practices across all technology stacks (backend, frontend, WordPress) to ensure:

- Consistent audit trails for compliance
- End-to-end distributed tracing
- Rapid incident response and debugging
- Cost-effective log management

Applies to: All production and staging environments across .NET, Node.js, Python, React, React Native, WordPress, and future platforms.

2. Core Principles

2.1 Trace-Centric Architecture

- All requests must generate a unique Trace ID that propagates across services
- Logs, metrics, and traces must be correlated via Trace ID + Span ID
- Use OpenTelemetry as the instrumentation standard

2.2 Structured Logging

- Never use plain text logs - always use structured JSON
- Include consistent metadata fields (see Section 4)
- Use semantic conventions for naming

2.3 Asynchronous Processing

- Log emission must not block the critical request path
- Use background workers/queues for log processing
- Target: <5ms overhead per request

2.4 Privacy by Design

- Never log sensitive data without explicit redaction
- Automatically mask: passwords, tokens, credit cards, SSNs, PII
- Implement field-level access controls

2.5 Standardized Metrics

All services must expose a standard set of metrics (Golden Signals) to provide a consistent, high-level view of system health.

3. Technology Stack & Architecture

3.1 Observability Data Flow

```
          Service (C#, Python, Node, React, WordPress)
                              ↓
     OTel Auto-Instrumentation + Manual Spans/Log Enrichment
                              ↓
  OpenTelemetry Collector (receivers → processors → exporters)
                              ↓
            SigNoz (metrics, logs, traces in one stack)
                              ↓
  Visualization & Analysis (UI dashboards, service maps, trace
                           explorers)
```

3.2 Required Components

Component	Purpose	Implementation

OpenTelemetry SDK	Instrumentation layer	Auto-instrumentation + custom spans
Trace Context Propagation	Distributed tracing	W3C Trace Context headers
Structured Logger	Log emission	Serilog (.NET), Winston (Node), Pino, structlog (Python)
OpenTelemetry Collector	Data pipeline	Centralized collection, processing, and routing
Observability Platform	Centralized monitoring	**SigNoz (Primary)**, Grafana Cloud (Alternative), Honeycomb.io (Alternative)
Background Queue	Async log processing	Channel-based queues (per stack)
Uptime Kuma	Synthetic monitoring	Internal network monitoring

3.3 Platform Selection & Rationale

Primary: SigNoz Cloud

Why SigNoz Cloud?

Technical Advantages:

1. **Unified Observability Stack**
   a. Single platform for logs, metrics, and traces (no need for separate tools)
   b. Native OpenTelemetry support with OTLP ingestion
   c. Built on ClickHouse for high-performance querying
   d. Automatic correlation between traces, logs, and metrics
2. **Zero Infrastructure Management**
   a. Fully managed ClickHouse clusters (no database tuning required)
   b. Automatic scaling based on ingestion volume
   c. Built-in high availability and disaster recovery
   d. No need to manage storage, backups, or upgrades
3. **Cost Efficiency**
   a. Transparent per-GB pricing: $0.30/GB for logs and traces
   b. No per-user pricing model - unlimited users
   c. Significantly cheaper than Grafana Cloud (89% savings) and Honeycomb
   d. Predictable costs with no hidden fees
4. **Developer-Friendly**

a. Modern UI with intuitive service maps and flame graphs
b. ClickHouse-powered fast queries (even for large datasets)
c. Query Builder for non-technical users
d. Native support for PromQL and ClickHouse SQL

5. **Feature Completeness**
   a. Distributed tracing with Jaeger-compatible UI
   b. Log aggregation with structured query language
   c. Metrics with Prometheus-compatible interface
   d. Alerts and notifications (Slack, PagerDuty, webhooks)
   e. Exception monitoring
   f. Service dependency mapping

Business Advantages:

1. **No Vendor Lock-In**
   a. Standard OpenTelemetry protocol
   b. Data portability (can export and migrate to self-hosted if needed)
   c. No proprietary agents or SDKs required
2. **Predictable Costs**
   a. Transparent per-GB pricing
   b. No surprise charges or user-based fees
   c. Free tier available for development and testing
3. **Active Development**
   a. Regular releases (monthly)
   b. Strong community support (10k+ GitHub stars)
   c. Enterprise support available
4. **Reduced Operational Overhead**
   a. No ClickHouse expertise required
   b. No infrastructure maintenance
   c. Automatic updates and security patches
   d. Focus on observability, not infrastructure management
5. **Quick Time to Value**
   a. Setup in minutes (vs. days for self-hosted)
   b. No capacity planning or scaling concerns
   c. Built-in monitoring and alerting

Alternative: SigNoz Self-Hosted

When to Use:

- Strict data sovereignty requirements
- Need full control over infrastructure
- Have in-house ClickHouse expertise
- Very high data volumes (>1TB/month where self-hosting becomes cheaper)
- Compliance requires on-premise deployment

Key Considerations:

- Requires ClickHouse management expertise
- Infrastructure maintenance overhead
- Manual scaling and capacity planning
- Cost: ~$250-500/month for infrastructure (100GB-500GB volume)

When to Use:

- Multi-cloud environments requiring centralized management
- Teams already invested in Grafana ecosystem
- Need for advanced Grafana features (Git Sync, SQL expressions)
- Compliance requirements (SOC 2 Type II certified)

Key Features:

- LGTM Stack (Loki, Grafana, Tempo, Mimir)
- Native OpenTelemetry support
- Free tier: 10K metrics, 50GB logs/traces
- Cost: $19/month base + $8/user + usage

Alternative: Honeycomb.io

When to Use:

- Complex debugging scenarios requiring advanced querying
- High-cardinality data analysis
- Teams requiring BubbleUp feature for anomaly detection

Key Features:

- Best-in-class query interface for high-cardinality data
- BubbleUp for automatic issue correlation
- Service Level Objectives (SLOs) built-in
- Cost: Starts at $0 (free tier), Pro at $35/user/month

3.4 Uptime Kuma for Internal Network Monitoring

Purpose: Complements cloud-based observability with internal infrastructure monitoring

Why Add Uptime Kuma:

- Self-hosted, open-source uptime monitoring
- Monitors HTTP(s), TCP, DNS, Ping, Docker containers
- Perfect for internal services behind firewalls
- Exposes Prometheus metrics at `/metrics` endpoint
- Built-in status pages for customer-facing uptime reporting
- 90+ notification integrations (Slack, Discord, Telegram, PagerDuty)
- Zero licensing cost

Note: For Production, an external 'Dead Man's Switch' must be configured to ping the Uptime Kuma instance. If Kuma goes down, the external service alerts the team.

Integration with SigNoz:

1. Deploy Uptime Kuma:

```
docker run -d --restart=always -p 3001:3001 \
  -v uptime-kuma:/app/data \
  --name uptime-kuma louislam/uptime-kuma:2
```

2. Configure OpenTelemetry Collector to scrape Uptime Kuma metrics:

```
receivers:
  prometheus:
    config:
      scrape_configs:
        - job_name: 'uptime-kuma'
          scrape_interval: 30s
          static_configs:
            - targets: ['uptime-kuma:3001']
          metrics_path: /metrics
          basic_auth:
            username: your_username
            password: your_api_key
```

3. Metrics exported by Uptime Kuma:
    a. `monitor_response_time` - Response latency
    b. `monitor_status` - Up/down status (1=up, 0=down)
    c. `monitor_cert_days_remaining` - SSL certificate expiration

When to Use Uptime Kuma:

- Internal infrastructure behind firewalls
- Development/staging environments
- Public customer-facing status pages
- Docker container health monitoring
- When you need monitoring from your internal network perspective

4. Mandatory Log Fields

4.1 Core Metadata (All Logs)

```
{
  "timestamp": "2025-11-25T10:30:45.123Z",
  "traceId": "4bf92f3577b34da6a3ce929d0e0e4736",
  "spanId": "00f067aa0ba902b7",
  "service": "payment-service",
  "environment": "production",
  "version": "1.2.3",
  "level": "INFO"
}
```

4.2 Request Logs (HTTP/API)

```
{
  // Core metadata (above) +
  "http": {
    "method": "POST",
    "path": "/api/v1/orders",
    "statusCode": 200,
    "duration": 145.67,
    "userAgent": "Mozilla/5.0..."
  },
  "user": {
    "id": "user_abc123",
    "tenantId": "tenant_xyz789",
    "role": "customer"
  },
  "client": {
    "ip": "203.0.113.42",
    "country": "PH"
  }
}
```

4.3 Error Logs

```
{
  // Core metadata +
  "error": {
    "type": "ValidationException",
    "message": "Invalid payment method",
    "stackTrace": "...",
    "code": "PAY_001"
  },
  "context": {
    "orderId": "ord_123",
    "amount": 1500.00
  }
}
```

4.4 Audit Logs (Compliance)

```
{
  // Core metadata +
  "audit": {
    "action": "USER_LOGIN",
    "actor": "user_abc123",
    "resource": "auth-service",
    "outcome": "SUCCESS",
    "changes": {
      "before": null,
      "after": { "status": "active" }
    }
  }
}
```

4.5 Frontend & RUM Logs

Safety Mechanism: The Frontend Logger must include a client-side rate limiter (e.g., max 50 events per minute per session) to prevent 'noisy neighbor' issues from buggy clients.

Page Views (including Core Web Vitals):

```
{
  // Core metadata +
  "event": "page_view",
  "page": {
    "url": "/checkout",
    "title": "Checkout",
    "referrer": "/cart"
  },
  "performance": {
    "fcp": 1200,
    "lcp": 2400,
    "fid": 50,
    "cls": 0.1,
    "ttfb": 300
  }
}
```

Frontend Errors:

```
{
  // Core metadata +
  "error": {
    "type": "TypeError",
    "message": "Cannot read property 'map' of undefined",
    "stackTrace": "...",
    "componentStack": "at ProductList\nat App"
  },
  "context": {
    "route": "/products",
    "userAction": "filter_clicked"
  }
}
```

User Actions:

```
{
  // Core metadata +
  "event": "user_action",
  "action": {
    "type": "button_click",
    "target": "checkout_button",
    "value": "Complete Purchase"
  },
  "session": {
    "id": "sess_abc123",
    "duration": 45000
  }
}
```

5. Naming Conventions

5.1 Service Names

- Format: `{domain}-{function}-service`
- Examples: `payment-processing-service`, `user-auth-service`
- Frontend: `web-app`, `mobile-app-ios`, `mobile-app-android`
- WordPress: `wordpress-cms`, `wordpress-api`

5.2 Span Names

- Format: `{HTTP_METHOD} {route}`
- Examples: `POST /api/orders`, `GET /api/users/{id}`
- Database: `db.query.users.select`, `db.query.orders.insert`
- External calls: `http.client.payment_gateway`, `http.client.email_service`

5.3 Log Levels

Level	Usage
TRACE	Fine-grained debugging (disabled in prod)
DEBUG	Detailed diagnostics (sample in prod)
INFO	Normal operations, business events
WARN	Recoverable errors, degraded performance
ERROR	Application errors requiring attention
FATAL	System failure, immediate action required

5.3.1 Dynamic Log Levels

The logging framework must support changing log levels at runtime without a service restart.

Recommendation: Support a per-request log level override via an HTTP header (e.g., `X-Log-Level: DEBUG`) for targeted production debugging.

5.4 Metric Conventions

Golden Signals: All services must emit metrics for Latency, Traffic, Errors, and Saturation.

Use standard OpenTelemetry semantic conventions where possible (e.g., `http.server.request.duration`).

Standard Metrics:

- `http.server.request.duration` - Request latency histogram
- `http.server.request.count` - Total request count
- `http.server.error.count` - Error count by status code
- `system.cpu.utilization` - CPU saturation
- `system.memory.utilization` - Memory saturation
- `db.client.operation.duration` - Database operation latency

6. Sensitive Data Handling

6.1 Automatic Redaction Rules

Never log in plain text:

- Passwords, API keys, tokens, secrets
- Credit card numbers (PAN)
- Social Security Numbers, Tax IDs
- Full names + DOB combinations
- Email addresses (in some regions)

Redaction Format:

```
{
  "creditCard": "[REDACTED:CREDIT_CARD]",
  "password": "[REDACTED:PASSWORD]",
  "ssn": "[REDACTED:SSN]",
  "email": "u***@example.com"
}
```

6.2 Implementation

Backend (.NET):

```
[Redact]
public string CreditCard { get; set; }

[LogMasked]
public string Email { get; set; } // Auto-masks: j***@email.com
```

Frontend (React/Node):

```
logger.info('Payment processed', {
  orderId: '123',
  amount: redactSensitive(paymentData.amount),
  card: maskCreditCard(paymentData.card)
});
```

7. Distributed Tracing

7.1 Trace Context Propagation

All HTTP requests must include W3C Trace Context headers:

```
traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01
```

Implementation:

Backend (.NET):

```
// Program.cs
services.AddOpenTelemetry()
  .WithTracing(builder => builder
    .AddAspNetCoreInstrumentation()
    .AddHttpClientInstrumentation()
    .AddOtlpExporter(options => {
      options.Endpoint = new Uri("http://signoz-otel-collector:4317");
      options.Protocol = OtlpExportProtocol.Grpc;
    }));
```

Frontend (React):

```
import { context, trace } from '@opentelemetry/api';

fetch('/api/orders', {
  headers: {
    'traceparent': getTraceParent(context.active())
  }
});
```

7.2 Custom Spans

When to create custom spans:

- External API calls (payment gateways, third-party services)
- Database queries (complex transactions)
- Expensive computations (>100ms)
- Business-critical operations (order processing, user registration)

Example (.NET):

```
using var activity = Activity.StartActivity("ProcessPayment");
activity?.SetTag("order.id", orderId);
activity?.SetTag("amount", amount);

try {
  var result = await _paymentGateway.Charge(amount);
  activity?.SetTag("transaction.id", result.TransactionId);
  return result;
} catch (Exception ex) {
  activity?.SetStatus(ActivityStatusCode.Error, ex.Message);
  throw;
}
```

8. OpenTelemetry Collector Configuration

8.1 Collector Architecture

The OpenTelemetry Collector acts as a centralized pipeline for all telemetry data, providing:

- Protocol translation (OTLP, Jaeger, Zipkin, Prometheus)
- Data enrichment and filtering
- Batching and retry logic
- Multi-backend support

8.2 Sample Collector Configuration

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

  prometheus:
    config:
      scrape_configs:
        - job_name: 'uptime-kuma'
          scrape_interval: 30s
          static_configs:
            - targets: ['uptime-kuma:3001']

processors:
  batch:
    timeout: 10s
    send_batch_size: 1024

  attributes:
    actions:
      - key: environment
        value: production
        action: insert
      - key: cluster
        value: us-central-1
        action: insert

  resource:
    attributes:
      - key: service.namespace
        value: ecommerce
        action: insert

  filter/drop-health-checks:
    traces:
      span:
        - 'attributes["http.target"] == "/health"'

exporters:
  otlp/signoz:
    endpoint: "signoz-otel-collector:4317"
    tls:
      insecure: true

  logging:
    loglevel: info

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch, attributes, resource, filter/drop-health-checks]
      exporters: [otlp/signoz]

    metrics:
      receivers: [otlp, prometheus]
      processors: [batch, attributes, resource]
      exporters: [otlp/signoz]

    logs:
      receivers: [otlp]
      processors: [batch, attributes, resource]
      exporters: [otlp/signoz]
```

8.3 Collector Deployment

Docker Compose:

```
version: '3.8'

services:
  otel-collector:
    image: otel/opentelemetry-collector-contrib:0.93.0
    command: ["--config=/etc/otel-collector-config.yaml"]
    volumes:
      - ./otel-collector-config.yaml:/etc/otel-collector-config.yaml
    ports:
      - "4317:4317"   # OTLP gRPC
      - "4318:4318"   # OTLP HTTP
      - "8888:8888"   # Prometheus metrics
      - "13133:13133" # health_check
    depends_on:
      - signoz
```

Kubernetes:

```
helm repo add open-telemetry https://open-telemetry.github.io/opentelemetry-helm-charts
helm install otel-collector open-telemetry/opentelemetry-collector \
  --set mode=deployment \
  --set config.receivers.otlp.protocols.grpc.endpoint="0.0.0.0:4317" \
  --set config.exporters.otlp.endpoint="signoz:4317"
```

9. Performance Targets

Metric	Target	Measurement
Log Overhead	<5ms per request	P95 latency increase
Queue Depth	<1000 messages	Background queue size
Log Processing Delay	<30 seconds	Time to searchable in SigNoz
Data Loss Rate	<0.01%	Failed log writes

9.1 Sampling Strategy

Production Environment:

- Success logs (2xx, 3xx): 10% sampling for high-traffic endpoints (>1000 req/min)
- Error logs (4xx, 5xx): 100% sampling (never sample errors)
- Slow requests (>2s): 100% sampling

Implementation:

```
bool shouldLog = statusCode >= 400
    || duration > 2000
    || Random.Shared.Next(100) < 10; // 10% sample rate
```

10. Retention & Storage Strategy

10.1 Data Tiers

Tier	Duration	Storage	Cost/GB	Use Case
Hot	0-7 days	SigNoz (ClickHouse)	$0 (self-hosted)	Real-time debugging
Warm	8-30 days	SigNoz (ClickHouse)	$0 (self-hosted)	Recent incidents

Cold	31-365 days	Blob Storage	$0.02	Compliance audits
Archive	1-7 years	Glacier/Archive	$0.004	Legal requirements

10.2 Lifecycle Policies

Automatic transitions:

- Day 7 → Retain in hot storage
- Day 30 → Export to Azure Blob/S3 (Parquet format)
- Day 365 → Move to Glacier/Archive tier
- Year 7 → Delete (unless legal hold)

11. Security & Compliance

11.1 Access Controls

Role	SigNoz Access	Database Logs	File Logs	Blob Storage
Developer	All logs (dev/staging)	Read-only	Read-only	None
DevOps	All logs (all envs)	Read/Write	Read/Write	Read/Write
Support	Errors only (prod)	None	None	None
Auditor	Audit logs only	Read-only	None	Read-only
Security Team	All logs	Read-only	Read-only	Read-only

11.2 Compliance Requirements

GDPR (if applicable):

- Right to erasure: Implement log deletion by user ID
- Data portability: Export logs in machine-readable format (JSON)
- Retention limits: Auto-delete after 2 years (unless justified)

HIPAA (if applicable):

- Encrypt logs at rest (AES-256) and in transit (TLS 1.3)
- Immutable audit trails for PHI access
- Business Associate Agreements (BAA) with vendors

SOC 2:

- Integrity verification: Hash each log entry
- Change detection: Alert on log tampering attempts
- Availability: 99.9% uptime for log ingestion

12. Alerting Strategy

12.1 Critical Alerts (Page Immediately)

Alert	Condition	Action
Tier 1 Error Spike (Payments, Auth)	>1% error rate for 5 min	Page on-call immediately
Tier 2 Error Spike (Reports, Avatars)	>5% error rate for 10 min	Page on-call
Tier 3 Error Spike (Internal Dev)	>10% error rate for 30 min	Slack Notification (No Page)
Service Down	No logs received for 10 min	Page + auto-restart
High Latency	P95 >5s for 10 min	Investigate performance
Security Breach	Multiple failed auth attempts	Lock account + alert security
Uptime Kuma Monitor Down	monitor_status == 0 for 2 consecutive checks	Page on-call + update status page

12.2 Warning Alerts (Slack/Email)

Alert	Condition	Action
High Queue Depth	>500 pending logs	Investigate backpressure
Elevated Errors	2-5% error rate	Monitor for escalation

Cost Overrun	>80% of monthly budget	Review sampling rates

13. Implementation Strategy & Tooling

To ensure consistency and reduce developer overhead, this standard shall be implemented through a suite of internal Configuration Libraries (not wrappers). These libraries provide a "paved road" for observability, ensuring that compliance is the default path while exposing native standard APIs to developers.

13.1 The Internal Configuration Library (Bootstrapper)

We will develop and maintain a set of lightweight internal packages (e.g., Company.Observability for .NET, @company/observability for Node.js).

Design Philosophy:

- Configuration, not Abstraction: The library will not wrap the OpenTelemetry API. It will merely configure it.
- No Vendor Lock-in: The library ensures all services point to the correct OTLP endpoint (SigNoz) and attach the correct Resource Attributes (Service Name, Version, Env).
- Native Usage: Developers will use standard logging interfaces (ILogger, console) and standard tracing APIs. They can consult official OpenTelemetry documentation rather than internal wikis.

13.2 Key Components

Backend Bootstrapper:

- Centralized Setup: A single extension method (e.g., AddCompanyObservability) that wires up the OpenTelemetry SDK.
- Resource Standardization: Automatically injects service.name, deployment.environment, and service.version from the build context.
- Auto-Instrumentation: Enables standard instrumentations (Http, Grpc, SqlClient) with sensible defaults.

- Log Enrichment: Configures the logging provider (Serilog/Winston) to export structured JSON to the OTel Collector automatically.

Frontend Bootstrapper:

- Trace Propagation: Configures the Fetch and XHR instrumentation to automatically inject traceparent headers—no manual developer intervention required.
- Rate Limiting: Enforces a client-side sampling rate (e.g., 10% of sessions) to prevent ingestion spikes from high-traffic clients.
- Error Boundary: Provides a plug-and-play React Error Boundary that sends stack traces to SigNoz with the correct session context.

13.3 Developer Workflow

- Install: Add the relevant internal package to the service.
- Initialize: Add the configuration line during application startup.
- Use: Use standard framework features. No custom logger objects are required.

Example (.NET):

```
// Program.cs
// The "Paved Road" - One line to configure OTel, Logging, and Metrics

builder.Services.AddCompanyObservability(options => {

    options.ServiceName = "payment-service";

    options.Environment = builder.Environment.EnvironmentName;

    // The library automatically handles OTLP export to SigNoz

});



// Controller.cs
public class PaymentController : ControllerBase

{

    // STANDARD Microsoft ILogger - No custom wrappers!
    private readonly ILogger<PaymentController> _logger;

    public PaymentController(ILogger<PaymentController> logger)
    {
        _logger = logger;
    }

    [HttpPost]
    public async Task<IActionResult> ProcessPayment(PaymentRequest request)
    {
        // The Bootstrapper ensures this log is formatted as JSON,
        // attached to the current TraceID, and sent to SigNoz.
        _logger.LogInformation("Processing payment for order {OrderId}", request.OrderId);

        // HTTP calls are automatically traced via Auto-Instrumentation
        await _paymentGateway.Charge(request.Amount);

        return Ok();
    }
}
```

14. Implementation Checklist

Backend Services (.NET/Node/Python/WordPress)

- [ ] Install OpenTelemetry SDK
- [ ] Configure trace context propagation
- [ ] Add structured logging library
- [ ] Implement audit middleware
- [ ] Create background log processor
- [ ] Add sensitive data redaction
- [ ] Configure OTLP exporter to Collector
- [ ] Define custom spans for critical operations
- [ ] Test trace correlation across services

Frontend Applications (React/React Native)

- [ ] Install OTel Web SDK
- [ ] Configure trace propagation in HTTP client
- [ ] Add browser/mobile error tracking
- [ ] Implement performance monitoring (Core Web Vitals)
- [ ] Redact sensitive form inputs
- [ ] Configure OTLP exporter to Collector
- [ ] Test trace correlation with backend
- [ ] Add user session tracking
- [ ] Implement RUM (Real User Monitoring)

Infrastructure

- [ ] Deploy SigNoz (Docker/Kubernetes)
- [ ] Deploy OpenTelemetry Collector
- [ ] Deploy Uptime Kuma for internal monitoring
- [ ] Configure Collector to forward to SigNoz
- [ ] Configure Collector to scrape Uptime Kuma metrics
- [ ] Set up RBAC and team access in SigNoz
- [ ] Configure data retention policies
- [ ] Create alerting rules (Critical: page, Warning: Slack)

- [ ] Set up log archival to blob storage
- [ ] Implement backup/disaster recovery
- [ ] Document runbooks for common scenarios
- [ ] Train team on SigNoz dashboard usage

15. Migration Strategy

Phase 1: Pilot (2 weeks)

- Implement in 1-2 non-critical services
- Deploy SigNoz and OpenTelemetry Collector
- Validate trace correlation
- Measure performance overhead
- Gather team feedback

Phase 2: Backend Rollout (4 weeks)

- Migrate all backend services
- Deploy Uptime Kuma for internal monitoring
- Establish baseline metrics (error rates, latency)
- Train backend engineers
- Document lessons learned

Phase 3: Frontend Rollout (3 weeks)

- Add OTel to web application
- Implement mobile app instrumentation (if applicable)
- Test end-to-end tracing (browser → backend → database)
- Optimize bundle size impact

Phase 4: Optimization (Ongoing)

- Fine-tune sampling rates
- Optimize cold storage costs
- Add custom dashboards and alerts

- Continuous improvement

16. Cost Estimation (November 2025)

16.1 SigNoz Cloud (Primary Recommendation)

Pricing Structure:

- **Base Plan:** $49/month (includes usage worth $49)
- **Overage Rates:**
  - Logs: $0.3/GB ingested (15 days retention)
  - Traces: $0.3/GB ingested (15 days retention)
  - Metrics: $0.1/million samples (1 month retention)

Monthly Breakdown (100GB/month ingestion):

Item	Calculation	Cost
Base Plan	Includes $49 worth of usage (~163GB logs/traces)	$49
Logs	100GB included in base plan	$0
Traces	50GB included in base plan	$0
Metrics	50K series ≈ 216M samples/month ≈ $22, but ~$10 covered in base	$12
Total		**~$61/month**

Alternative Calculation (if exceeding base plan): If your usage exceeds $49 worth:

- Logs: 100GB × $0.3/GB = $30
- Traces: 50GB × $0.3/GB = $15
- Metrics: 216M samples × $0.1/M = $22
- Total would be: $67/month (but base plan covers first $49, so ~$67/month)

Key Benefits:

- **Unlimited users** (no per-user fees)
- **Unlimited hosts** (no per-host fees)

- Fully managed ClickHouse (no database expertise needed)
- Automatic scaling and high availability
- No infrastructure maintenance
- SOC2 Type II & HIPAA compliant
- 15-day retention included (logs/traces)
- Support via in-product chat, email, and Slack
- Data centers in US, EU, and India

Why We Chose SigNoz Cloud Over Self-Hosted:

- **Zero ClickHouse Management:** No need for database tuning, scaling, or maintenance
- **Faster Time to Value:** Setup in minutes vs. days
- **Cost Effective:** At $49-67/month for 100GB, significantly cheaper than alternatives
- **Built-in HA & Disaster Recovery:** No need to architect and maintain redundancy
- **Automatic Updates:** Always on latest version without manual upgrades
- **Focus on Observability:** Team can focus on using data, not managing databases

Free Tier Available:

- Self-hosted Community Edition (100% free, unlimited usage)
- Perfect for development and staging environments

16.2 SigNoz Self-Hosted (For Reference)

Cost Structure:

- **Licensing:** $0 (100% free, open source)
- **Infrastructure Costs:** Variable based on your setup

Estimated Monthly Infrastructure Costs (100GB/month ingestion):

Component	Specifications	Estimated Cost Range
Compute (VM/Kubernetes)	4-8 vCPU, 16-32GB RAM	$100-200/month
Storage (SSD)	500GB-1TB	$30-80/month
Backup Storage	500GB-1TB	$10-25/month
Bandwidth/Egress	Varies by provider	$20-50/month

Total		**$160-355/month**

Note: Costs vary significantly based on:

- Cloud provider (AWS, GCP, Azure, DigitalOcean, Hetzner)
- Region selection
- Reserved vs. on-demand instances
- Storage tier selection
- Data transfer patterns

Additional Considerations:

- ClickHouse management expertise required
- Manual scaling and capacity planning
- Backup and disaster recovery setup
- Infrastructure maintenance burden
- Security patching and updates

When Self-Hosted Makes Sense:

- Data volumes >2-3TB/month (cost crossover point)
- Strict data sovereignty requirements (on-premise only)
- Existing ClickHouse team and expertise
- Regulatory compliance requires self-hosting
- Already have spare infrastructure capacity

16.3 SigNoz Cloud vs Self-Hosted Decision Matrix

Scenario: 10 Hosts, 500GB Logs, 1TB Metrics, 10 users

Item	Details	Cost
Base Subscription	$49/month includes ~163GB logs/traces or ~490M metric samples	**$49**
Logs	500GB – 163GB included = 337GB × $0.30/GB	**~$101**
Metrics	1TB ≈ ~1B samples – 490M included = 510M × $0.10 per million	**~$51**
Hosts/Users	Unlimited hosts and seats included	**$0**

Total		**~$201/month**

Item	Details	Cost
Licensing	Open source, no license fees	**$0**
Infrastructure	100 hosts + storage for 1.5TB telemetry/month	**~$300–500/month** (VMs, storage, ops)
Ops/Maintenance	Ongoing ClickHouse tuning, scaling, backups	**Time + expertise cost**
Total		**~$300–500/month** (infra only)

Recommendation: Start with SigNoz Cloud. It's more cost-effective at typical scales and eliminates operational overhead. Consider self-hosted only if:

- Data volumes consistently exceed 2-3TB/month
- Regulatory requirements mandate on-premise deployment
- You already have ClickHouse expertise and spare infrastructure

16.4 Grafana Cloud (Alternative)

Monthly Breakdown (10 Hosts, 500GB Logs, 1TB Metrics, 10 users):

Item	Details	Cost
Metrics	1TB ≈ ~50K series (40K above included 10K) × $6.50/1K	**$260**
Logs	500GB – 50GB included = 450GB × $0.50/GB	**$225**
Traces	50GB included	**$0**
Visualization Users	Assume 10 users – 3 included → 7 × $8	**$56**

Hosts (Application Observability)	10 × 720 = 7,200 host hours – 2,232 included = 4,968 × $0.04/hr	**~$199**
Total		**~$740/month**

Note: Previous calculations were based on older pricing. Current pricing is actually lower than previously stated, but still significantly more expensive than SigNoz Cloud.

Cost vs. SigNoz Cloud: $341 vs $49-67 = 5-7x more expensive

When Grafana Cloud Makes Sense:

- Already using Grafana dashboards extensively
- Need specific Grafana Enterprise plugins
- Existing team expertise with Grafana ecosystem
- Multi-cloud visualization requirements

16.4 Honeycomb.io (Alternative)

Monthly Breakdown (100GB/month, 10 users):

Item	Details	Cost
Base Subscription	Pro plan covers up to 100M events/month (~100GB telemetry)	**$130**
Event Volume	500GB logs + 1TB metrics ≈ ~1.5B events	**15 × $130 = $1,950**
User Seats	Unlimited	**$0**
Total		**~$1,950/month**

16.5 Uptime Kuma (Internal Monitoring)

Item	Cost
Infrastructure (single VM)	$10/month
Self-hosted licensing	$0
Total	**$10/month**

16.6 Cost Comparison Summary

Platform	Monthly Cost	Savings vs. Grafana	Notes
SigNoz (Self-Hosted)	**~$300–500**	**~32–59% cheaper**	Best for full control, infra + ops overhead
SigNoz Cloud	**~$201**	**~73% cheaper**	Best for balanced needs, no host/user charges
Grafana Cloud Pro	**~$740**	**Baseline**	Host-based pricing dominates even at small scale
Honeycomb.io Pro	**~$1,950**	**~163% more expensive**	Best for high-cardinality, event-based pricing
Uptime Kuma	**~$10**	**N/A**	Internal monitoring only, not full observability

16.7 Cost Optimization Strategies

1. **Self-Host SigNoz (40–60% savings)**
   a. Use reserved cloud instances for compute
   b. Optimize ClickHouse storage with compression
2. **Log Sampling (30% savings)**
   a. Sample 10% of success logs in high-traffic endpoints
   b. Retain 100% of errors and slow requests
3. **Cold Storage Lifecycle (20% savings)**
   a. Move logs older than 30 days to Azure Blob/S3
   b. Cost reduction: ClickHouse storage → $0.02/GB (cold)
4. **Smart Retention Policies**
   a. Keep only error logs beyond 30 days
   b. Aggregate metrics to lower resolution after 7 days
5. **Uptime Kuma vs Cloud Monitoring**
   a. Use Uptime Kuma ($10/month) for internal services
   b. Reserve cloud synthetic monitoring for critical public endpoints

16.8 Projected Costs by Team Size

SigNoz Cloud:

Team Size	Data Volume	Monthly Cost	Annual Cost	Notes
5 users	50GB	**$49**	**$588**	Within base plan
10 users	100GB	**$49**	**$588**	Still within base plan (covers ~163GB)
25 users	250GB	**~$79**	**~$948**	Base + ~87GB overage ($0.30/GB ≈ $26)
50 users	500GB	**~$199**	**~$2,388**	Base + ~337GB overage ($0.30/GB ≈ $101) + metrics
100 users	1TB	**~$389**	**~$4,668**	Base + ~837GB overage ($0.30/GB ≈ $251) + metrics; consider Enterprise tier

Cost Calculation Formula:

- If total usage cost ≤ $49: Pay only $49/month
- If total usage cost > $49: Pay the actual usage cost
- Logs/Traces: $0.3/GB
- Metrics: $0.1/million samples

SigNoz Self-Hosted:

Team Size	Estimated Monthly Cost	Annual Cost	Notes
5 users	**$160–250**	**$1,920–3,000**	Basic infrastructure, $0 licensing
10 users	**$160–300**	**$1,920–3,600**	May need better specs (CPU/RAM scaling)
25 users	**$250–400**	**$3,000–4,800**	Requires scaling infrastructure (larger VM/storage)
50 users	**$350–550**	**$4,200–6,600**	Larger infrastructure needed, higher ops overhead
100 users	**$500–800**	**$6,000–9,600**	Enterprise-grade setup, HA + ops expertise

Important: These are infrastructure cost estimates only. SigNoz self-hosted is **free (open source)** with $0 licensing fees. Actual costs vary based on cloud provider, region, and specs.

Grafana Cloud (For Comparison):

Team Size	Data Volume	Monthly Cost	Annual Cost	Cost vs. SigNoz
5 users	50GB	**~$186**	**~$2,232**	**3.8× more expensive**
10 users	100GB	**~$341**	**~$4,092**	**5.1–7× more expensive**
25 users	250GB	**~$461**	**~$5,532**	**4.2–9.4× more expensive**
50 users	500GB	**~$581**	**~$6,972**	**2.9–11.9× more expensive**

Note: Grafana costs here is based on metrics ($6.50/1K series), logs ($0.50/GB), and users ($8/user). Actual costs vary based on metrics cardinality and host.

SigNoz Startup Program (If Eligible):

Team Size	Data Volume	Monthly Cost	Annual Cost
5 users	50GB	$24.50	$294
10 users	100GB	$24.50-34	$294-408
25 users	250GB	$54.50	$654

16.9 ROI Analysis

Traditional Approach (No Unified Observability):

- Multiple disconnected tools
- Manual log aggregation
- Difficult troubleshooting
- Higher MTTR (Mean Time To Resolution)
- Estimated cost of downtime: **$5,000–50,000/incident**

With SigNoz Cloud:

- **Total Cost:** ~$201/month (**$2,412/year**)
- **Reduced MTTR:** 70% improvement (from hours to minutes)

- **Prevented incidents:** Proactive alerting
- **Developer productivity:** 20% time savings on debugging
- **ROI:** Pays for itself with **1 prevented incident per year**

Cost Comparison vs. Traditional Tools (10 Hosts):

Platform	Monthly Cost	Cost vs. SigNoz
SigNoz Cloud	~$201	Baseline
Grafana Cloud Pro	~$740	**3.7× more**
New Relic	~$384	**1.9× more**
Datadog	~$1,470	**7.3× more**

Annual Savings:

- **vs. Grafana Cloud:** ~$6,500/year saved
- **vs. New Relic:** ~$2,200/year saved
- **vs. Datadog:** ~$15,200/year saved

For Eligible Startups (50% off SigNoz Cloud):

- **Annual Cost:** ~$1,206/year
- Even more compelling ROI for early-stage companies

16.10 Enterprise Tier

For organizations with larger scale or specific requirements:

Enterprise Cloud:

- Starts at $4,000/month (includes ingestion usage till $4,000)
- Volume discounts available
- Annual contracts with better rates

Enterprise Features:

- HIPAA & BAA agreements
- Dedicated Slack, email & in-product support
- Guided migration support

- Ongoing professional services
- Team training
- SLA with downtime developer pairing
- Advanced RBAC with custom roles
- AWS Private Link
- Audit logs
- Multi-tenancy

Bring Your Own Cloud (BYOC):

- SigNoz manages infrastructure in your cloud
- Full data sovereignty
- Custom pricing

When to Consider Enterprise:

- Data volumes consistently >1TB/month
- Compliance requires BAA or specific certifications
- Need dedicated support and SLAs
- Require advanced RBAC and multi-tenancy
- Want migration assistance from other platforms

17. Success Metrics

17.1 Technical KPIs

- **Mean Time to Detection (MTTD):** <5 minutes
- **Mean Time to Resolution (MTTR):** <30 minutes
- **Log Coverage:** >95% of requests traced
- **Data Loss Rate:** <0.01%
- **Query Performance:** <3 seconds for 90% of queries

17.2 Business KPIs

- **Audit Query Time:** <10 seconds for 30-day range
- **Compliance Violations:** 0 incidents

- **Cost per GB:** <$0.35 (including storage)
- **Developer Satisfaction:** >4/5 rating
- **Incident Prevention:** Proactive detection before customer impact

17.3 Operational KPIs

- **Service Uptime:** 99.9%
- **Alert Accuracy:** >90% (low false positives)
- **Dashboard Usage:** >80% of team using regularly
- **Time Saved on Debugging:** >20% reduction

18. Governance

18.1 Ownership

Primary Owner:

- 1 Senior Engineer (Observability Lead)
- Responsible for: Standard updates, training, best practices

Backup Owner:

- 1 DevOps/SRE Engineer
- Responsible for: Infrastructure, alerting, incident response

Stakeholders:

- Engineering Team Leads
- Security Team
- Compliance Officer (if applicable)

18.2 Change Management Process

Major Changes (affecting architecture, costs, or compliance):

- Proposal document required
- Synchronous review meeting (48hr notice)

- Approval from 2+ team leads
- Documentation update
- Team training session

Minor Changes (configuration, thresholds, dashboards):

- Pull request with clear description
- Review by 2 engineers
- Merge after approval
- Update runbooks if needed

Emergency Changes (incidents, security):

- Immediate implementation allowed
- Retrospective review within 48 hours
- Document learnings and update standard

18.3 Review Cadence

- **Weekly:** Review critical alerts and false positives
- **Monthly:** Cost review and optimization
- **Quarterly:** Standard retrospective and improvements
- **Annually:** Platform evaluation and vendor review

18.4 Training & Onboarding

New Engineers:

- Observability onboarding session (1 hour)
- Hands-on workshop with sample scenarios
- Access to runbooks and documentation
- Shadow on-call rotation

Ongoing Training:

- Monthly "Observability Office Hours"
- Quarterly deep-dive sessions on advanced features
- Share interesting traces/incidents in team meetings

19. Appendices

19.1 OpenTelemetry (.NET Implementation)

Official Documentation:

- OpenTelemetry .NET Documentation
- Getting Started with OpenTelemetry .NET
- Microsoft Learn: .NET Observability with OpenTelemetry
- Microsoft Learn: Add Distributed Tracing Instrumentation
- GitHub: OpenTelemetry .NET Client Repository

Tutorials & Best Practices:

- Medium: Practical Guide to Implementing Telemetry in .NET (Sep 2024)
- Stackify: OpenTelemetry for .NET Engineers Guide
- Getting Started with OpenTelemetry and Distributed Tracing in .NET (Apr 2024)

19.2 SigNoz Documentation

Official Resources:

- SigNoz Official Website
- SigNoz Documentation
- SigNoz Pricing
- GitHub: SigNoz Repository
- SigNoz Cloud

Integration Guides:

- Instrument .NET Application with OpenTelemetry
- Instrument Node.js Application
- Instrument Python Application
- Instrument React Application
- Instrument WordPress

Tutorials:

- SigNoz Blog: OpenTelemetry Collector Complete Guide
- SigNoz Blog: Distributed Tracing in Microservices
- SigNoz Blog: Log Management
- SigNoz Blog: Pricing Comparison vs Datadog, New Relic, Grafana

19.3 OpenTelemetry Collector

Official Documentation:

- OpenTelemetry Collector Documentation
- Collector Configuration
- GitHub: OpenTelemetry Collector Contrib

Deployment Guides:

- Deploy with Docker
- Deploy with Kubernetes
- SigNoz: OpenTelemetry Collector Setup

19.4 Serilog (Structured Logging for .NET)

Official Resources:

- Serilog Official Website
- GitHub: Serilog Repository

Best Practices:

- 5 Serilog Best Practices for Better Structured Logging (Dec 2023)
- Code Maze: Best Practices for Logging with Serilog
- Ben Foster: Serilog Best Practices

19.5 Winston (Node.js Logging)

Official & Community Resources:

- Better Stack: Complete Guide to Winston Logging in Node.js
- Dash0: Mastering Winston for Production Logging in Node.js
- Stackify: Winston Logger Ultimate Tutorial

19.6 React Error Boundaries & Frontend Logging

Official React Documentation:

- React: Error Boundaries Documentation

Best Practices:

- React Error Handling and Logging Best Practices (Mar 2025)
- Sentry: Guide to Error & Exception Handling in React (Jun 2025)
- LogRocket: React Error Handling with react-error-boundary (Jul 2024)

19.7 W3C Trace Context & Distributed Tracing

Official W3C Specifications:

- W3C Trace Context Specification
- W3C Trace Context Level 2
- GitHub: W3C Trace Context Repository

Implementation Guides:

- OpenTelemetry: Propagation Guide
- Google Cloud: Trace Context Documentation

19.8 Sensitive Data Redaction & PII Protection

GDPR & Compliance:

- Skyflow: How to Keep Sensitive Data Out of Your Logs (Feb 2025)
- Datadog: Redact Sensitive Data from Logs On-Prem

Industry-Specific:

- OWASP Logging Cheat Sheet
- GDPR Official Text
- HIPAA Security Rule

19.9 Uptime Kuma Integration

Uptime Kuma Resources:

- GitHub: Uptime Kuma Official Repository
- GitHub Wiki: Uptime Kuma Prometheus Integration
- Uptime Kuma Documentation

19.10 Additional Resources

OpenTelemetry Community:

- OpenTelemetry Main Site
- OpenTelemetry Demo Applications
- CNCF OpenTelemetry Project

SigNoz Community:

- SigNoz Slack Community
- SigNoz GitHub Discussions
- SigNoz Blog

ClickHouse Resources (For Reference):

- ClickHouse Official Documentation
- ClickHouse Best Practices

20. Glossary

Term	Definition
OTLP	OpenTelemetry Protocol - Standard protocol for telemetry data
Span	A single unit of work in a distributed system
Trace	End-to-end journey of a request across multiple services
Golden Signals	Four key metrics: Latency, Traffic, Errors, Saturation
Core Web Vitals	Google's metrics for page performance (LCP, FID, CLS)

MTTD	Mean Time to Detection - Time to detect an issue
MTTR	Mean Time to Resolution - Time to resolve an issue
RUM	Real User Monitoring - Frontend performance tracking
PII	Personally Identifiable Information
Cardinality	Number of unique values for a given attribute
ClickHouse	Column-oriented database optimized for analytics
W3C Trace Context	Standard for propagating trace information across services
Semantic Conventions	Agreed-upon naming standards for telemetry attributes
APM	Application Performance Monitoring
SLO	Service Level Objective - Target for service performance
SLA	Service Level Agreement - Contractual performance guarantee
RBAC	Role-Based Access Control
BAA	Business Associate Agreement (for HIPAA compliance)

21. Quick Reference Guide

Common Commands

OpenTelemetry Collector:

```
# Start collector
docker run -d --name otel-collector \
  -p 4317:4317 -p 4318:4318 \
  otel/opentelemetry-collector-contrib:0.93.0

# View collector logs
docker logs -f otel-collector

# Restart collector
docker restart otel-collector
```

SigNoz:

```
# Access SigNoz Cloud
https://your-org.signoz.cloud

# API endpoint for data ingestion
https://ingest.{region}.signoz.cloud:443
```

Uptime Kuma:

```
# Start Uptime Kuma
docker run -d --restart=always -p 3001:3001 \
  -v uptime-kuma:/app/data \
  --name uptime-kuma louislam/uptime-kuma:2

# Access Uptime Kuma
http://localhost:3001
```

Troubleshooting Checklist

Logs not appearing in SigNoz:

- [ ] Verify OTLP collector endpoint is correct
- [ ] Check collector is running (`docker ps`)
- [ ] Verify network connectivity to SigNoz Cloud
- [ ] Check API key/authentication is valid
- [ ] Review collector logs for errors
- [ ] Confirm log format is JSON structured

Traces not correlated:

- [ ] Verify W3C Trace Context headers are present
- [ ] Check `traceparent` header format
- [ ] Ensure all services propagate trace context
- [ ] Verify trace ID is consistent across services
- [ ] Check OpenTelemetry SDK is properly initialized

High costs:

- [ ] Review sampling rates (increase sampling for success logs)
- [ ] Check for duplicate log entries
- [ ] Verify log level is appropriate (not DEBUG in prod)
- [ ] Review metric cardinality
- [ ] Consider archiving old data to blob storage

Slow queries in SigNoz:

- [ ] Add indexes to frequently queried fields
- [ ] Reduce query time range
- [ ] Use filters to reduce data scanned
- [ ] Check if retention period is too long
- [ ] Contact SigNoz support for optimization

Emergency Contacts

On-Call Rotation:

- Primary: [Your Team's On-Call Schedule]
- Backup: [Backup Schedule]
- Escalation: [Manager Contact]

Vendor Support:

- SigNoz Support: support@signoz.io
- SigNoz Slack: https://signoz.io/slack
- Emergency Hotline: [If applicable for Enterprise tier]

Last Updated: November 25, 2025