Phase 1H

Locked and folded into Phase-1 (1H). Here's the minimal, fast, reproducible CI you can drop in now.

Decisions (from your research → applied)

- Runner: Ubuntu, Node 22 LTS (single version for speed).
- PM: pnpm with store caching via actions/setup-node (cache: pnpm).
- Order: install → type-check first → tests (skip coverage in Phase-1).
- Security: actions pinned by SHA, least-privilege permissions: contents: read.
- Speed: concurrency.cancel-in-progress: true.
- No matrix (add later).

Add: .github/workflows/ci.yml

```
name: CI
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
permissions:
  contents: read
concurrency:
  group: ci-${{ github.ref }}
  cancel-in-progress: true
jobs:
  verify:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@a5ac7e51b41094c7d46d46d10a6476e22ff0f611 # v4 pinned
      - uses: pnpm/action-setup@fe02b34f77f8bc703788d5817da081398fad5dd2 # v4 pinned
        with:
          version: 10
      - uses: actions/setup-node@60edb5dd545a775178fbb8726080b81d56d67170 # v4 pinned
        with:
          node-version: 22
          cache: pnpm
      - run: pnpm install --frozen-lockfile
      - run: pnpm run type-check
      - run: pnpm test -- --run
```

Package scripts (ensure these exist)

```
{
  "scripts": {
    "type-check": "tsc --noEmit",
    "test": "vitest"
  }
}
```

Make it a required check

Set the required status to the job name "verify" (workflow CI) in your branch protection so merges wait on typecheck+tests.

If you want an npm variant or a Node 20 + 22 matrix later, say the word and I'll spit out the alt workflow.

Awesome — here are both variants, ready to drop in. Pick one (or keep both as separate workflows).

A) npm (single Node 22, fastest)

.github/workflows/ci-npm.yml

```
name: CI (npm)
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
permissions:
  contents: read
concurrency:
  group: ci-npm-${{ github.ref }}
  cancel-in-progress: true
jobs:
  verify:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@a5ac7e51b41094c7d46d46d10a6476e22ff0f611 # v4 pinned
      - uses: actions/setup-node@60edb5dd545a775178fbb8726080b81d56d67170 # v4 pinned
        with:
          node-version: 22
          cache: npm
          # cache-dependency-path: '**/package-lock.json' # uncomment for monorepos
      - run: npm ci
      - run: npm run type-check # tsc --noEmit
      - run: npm test -- --run   # vitest non-watch
```

B) npm + Node 20 & 22 matrix (broader coverage)

.github/workflows/ci-npm-matrix.yml

```
name: CI (npm matrix)
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
permissions:
  contents: read
concurrency:
  group: ci-npm-matrix-${{ github.ref }}
  cancel-in-progress: true
jobs:
  verify:
    runs-on: ubuntu-latest
    strategy:
      fail-fast: true
      matrix:
        node: [20, 22]
    name: verify (node: ${{ matrix.node }})
    steps:
      - uses: actions/checkout@a5ac7e51b41094c7d46d46d10a6476e22ff0f611
      - uses: actions/setup-node@60edb5dd545a775178fbb8726080b81d56d67170
        with:
          node-version: ${{ matrix.node }}
          cache: npm
          # cache-dependency-path: '**/package-lock.json' # uncomment for monorepos
      - run: npm ci
      - run: npm run type-check
      - run: npm test -- --run
```

package.json scripts (ensure)

```
{
  "scripts": {
    "type-check": "tsc --noEmit",
    "test": "vitest"
  }
}
```

Note on required checks

- Single-version workflow: require verify.

- Matrix workflow: GitHub creates separate checks per job (e.g., verify (node: 20), verify (node: 22)). Mark both as required in branch protection.

Phase 1I

Awesome research drop. Let’s lock it in with a tiny PR that adds a stable log schema + trace/correlation stubs and a JSONL test. It’s small, PoC-safe, and won’t churn later.

🧩 Tiny PR diff — “1I: minimal logging fields + telemetry stub”

```
diff --git a/src/telemetry/schema.ts b/src/telemetry/schema.ts
new file mode 100644
index 0000000..a1b2c3d
--- /dev/null
+++ b/src/telemetry/schema.ts
@@ -0,0 +1,63 @@
+// Stable, OTEL-aligned minimal log record (won't churn later).
+// Maps to: Timestamp, Severity{Text/Number}, Body(msg), Attributes, Resource,
+// TraceId, SpanId, TraceFlags, EventName, plus app-level correlationId.
+export type TelemetryLog = {
+  /** ISO 8601 in UTC (e.g. 2025-10-28T13:37:42.123Z) */
+  timestamp: string;
+  /** "TRACE"|"DEBUG"|"INFO"|"WARN"|"ERROR"|"FATAL" */
+  level: string;
+  /** 1..24 per OTEL severity mapping */
+  severityNumber: number;
+  /** Human message (kept small and stable) */
+  msg: string;
+  /** Optional event name for quick faceting (e.g. "agent.step") */
+  eventName?: string;
+  /** Free-form attributes (keep keys stable; redact secrets upstream) */
+  attributes?: Record<string, unknown>;
+  /** OTEL resource-ish metadata */
+  resource: {
+    'service.name': string;
+    'service.version'?: string;
+    'deployment.environment'?: string;
+    'host.name'?: string;
+  };
+  /** W3C trace context bits (lightweight now, fuller later) */
+  traceId?: string;
+  spanId?: string;
+  traceFlags?: string; // "00" | "01"
+  /** App-level correlation when full tracing isn’t present */
+  correlationId?: string;
+};
+
+export const severityToNumber = (level: string): number => {
+  switch (level.toUpperCase()) {
+    case 'TRACE': return 1;
+    case 'DEBUG': return 5;
+    case 'INFO':  return 9;
+    case 'WARN':  return 13;
+    case 'ERROR': return 17;
+    case 'FATAL': return 21;
+    default:      return 9;
+  }
+};
diff --git a/src/logger.ts b/src/logger.ts
```

```
index 4d5e6f7..9abc012 100644
--- a/src/logger.ts
+++ b/src/logger.ts
@@ -1,20 +1,67 @@
-import pino from 'pino';
+import pino from 'pino';
+import { trace } from '@opentelemetry/api';
+import { hostname } from 'node:os';
+import { randomUUID } from 'node:crypto';
+import { severityToNumber, type TelemetryLog } from './telemetry/schema.js';

-// existing minimal logger from 1G…
-export const logger = pino({
-  level: process.env.LOG_LEVEL || 'info',
-  timestamp: pino.stdTimeFunctions.isoTime,
-  formatters: {
-    level(label) { return { level: label.toUpperCase() }; }
-  },
-  redact: {
-    paths: ['password', 'req.headers.authorization', '*.apiKey', '*.token'],
-    censor: '[REDACTED]',
-    remove: false
-  }
-});
+const SERVICE_NAME = process.env.SERVICE_NAME ?? 'orchestration-build';
+const SERVICE_VERSION = process.env.SERVICE_VERSION ?? '0.0.0';
+const ENV = process.env.NODE_ENV ?? 'development';
+
+let _correlationId: string | undefined;
+export const setCorrelationId = (id: string) => { _correlationId = id; };
+export const getCorrelationId = () => (_correlationId ??= randomUUID());
+
+export const logger = pino(
+  {
+    level: process.env.LOG_LEVEL || 'info',
+    timestamp: pino.stdTimeFunctions.isoTime, // ISO UTC
+    formatters: {
+      level(label) { return { level: label.toUpperCase() }; },
+      bindings(b) { return { pid: b.pid, hostname: b.hostname }; }
+    },
+    // Fail-closed redaction (deterministic)
+    redact: {
+      paths: [
+        'password',
+        'req.headers.authorization',
+        'req.headers.cookie',
+        '*.apiKey',
+        '*.token',
+        'user.email',
+        'payment.card.number'
+      ],
+      censor: '[REDACTED]',
+      remove: false
+    },
+    base: {
+      resource: {
+        'service.name': SERVICE_NAME,
+        'service.version': SERVICE_VERSION,
+        'deployment.environment': ENV,
+       'host.name': hostname()
+     }
+   },
+   // Light OTEL/W3C bridge + correlation
+   mixin() {
+     const span = trace.getActiveSpan();
+     const ctx = span?.spanContext();
+     return {
+       traceId: ctx?.traceId,
+       spanId: ctx?.spanId,
+       traceFlags: ctx ? ctx.traceFlags.toString(16).padStart(2, '0') : undefined,
+       correlationId: getCorrelationId()
+     } as Partial<TelemetryLog>;
+   }
+  }
+);
+
+// Helper to emit a fully shaped record (stable keys)
+export function logStable(level: TelemetryLog['level'], msg: string, extras: Partial<TelemetryLog> = {}) {
+  const entry: TelemetryLog = {
+   timestamp: new Date().toISOString(),
+   level,
+   severityNumber: severityToNumber(level),
+   msg,
+   resource: (logger as any).bindings()?.resource ?? { 'service.name': SERVICE_NAME },
+   ...extras
+  };
+  (logger as any)[level.toLowerCase()]?.(entry);
+}

 export default logger;
diff --git a/test/logger.telemetry.test.ts b/test/logger.telemetry.test.ts
new file mode 100644
index 0000000..c0ffee1
--- /dev/null
+++ b/test/logger.telemetry.test.ts
@@ -0,0 +1,63 @@
+import { describe, it, expect } from 'vitest';
+import pino from 'pino';
+import { Writable } from 'node:stream';
+import { logger as appLogger, logStable, setCorrelationId } from '../src/logger.js';
+
+class Capture extends Writable {
+  chunks: string[] = [];
+  _write(chunk: any, _enc: any, cb: any) { this.chunks.push(String(chunk)); cb(); }
+}
+
+describe('telemetry JSONL', () => {
+  it('emits stable JSONL with minimal OTEL fields + redaction', async () => {
+   const sink = new Capture();
+   // Local logger mirroring app config but writing to in-memory sink
+   const local = pino((appLogger as any).opts, sink as any);
+   setCorrelationId('11111111-1111-4111-8111-111111111111');
+   local.info({
+     timestamp: new Date('2025-10-28T12:00:00.000Z').toISOString(),
+     level: 'INFO',
+     severityNumber: 9,
+     msg: 'hello',
+     attributes: { 'req.headers.authorization': 'Bearer SECRET' },
+      resource: { 'service.name': 'test' }
+    });
+    await new Promise(r => setImmediate(r));
+    expect(sink.chunks.length).toBeGreaterThan(0);
+    const line = sink.chunks[0];
+    expect(line.endsWith('\n')).toBe(true); // JSON Lines contract
+    const obj = JSON.parse(line);
+    expect(obj.level).toBe('INFO');
+    expect(obj.msg).toBe('hello');
+    expect(obj.severityNumber).toBe(9);
+    expect(obj.resource['service.name']).toBe('test');
+    expect(obj.correlationId).toBeDefined();
+    // redaction applied
+    expect(obj.attributes['req.headers.authorization']).toBe('[REDACTED]');
+  });
+});
diff --git a/docs/logging.md b/docs/logging.md
new file mode 100644
index 0000000..b00b1e5
--- /dev/null
+++ b/docs/logging.md
@@ -0,0 +1,33 @@
+# Logging & Telemetry (Phase 1I)
+
+- **Format:** JSON Lines (one JSON object per line, UTF-8, `\\n` terminated).
+- **Stable fields:** `timestamp`, `level`, `severityNumber`, `msg`, `eventName?`,
+  `attributes?`, `resource`, `traceId?`, `spanId?`, `traceFlags?`, `correlationId?`.
+- **Resource defaults:** `service.name`, `service.version`, `deployment.environment`, `host.name`.
+- **Redaction:** Allowlist/paths with deterministic `[REDACTED]`.
+- **Tracing:** If W3C context present, we include `traceId/spanId/traceFlags`. Otherwise we still emit
`correlationId`.
+
+### Sample
+```json
+{"timestamp":"2025-10-28T12:00:00.000Z","level":"INFO","severityNumber":9,"msg":"hello","resource":{"servic
e.name":"orchestration-build"},"correlationId":"..."}
+```
diff --git a/package.json b/package.json
index 1e2e3f4..5a6b7c8 100644
--- a/package.json
+++ b/package.json
@@ -7,6 +7,8 @@
   "scripts": {
     "build": "tsc -p tsconfig.json",
     "test": "vitest run --reporter=dot",
+    "log:sample": "node -e \"console.log(JSON.stringify({timestamp:new
Date().toISOString(),level:'INFO',severityNumber:9,msg:'hello',resource:{'service.name':'orchestration-build'}})+'\\
n')\"",
+    "lint:tsc": "tsc --noEmit"
   }
 }
```

What this gives you

- Stable schema (OTEL-aligned) now, so later phases won't break stored logs.
- Lightweight trace/correlation that just works today (and upgrades cleanly to full OTEL later).
- Deterministic redaction (same in mocks & prod).
- JSONL test proving: one-line JSON, stable keys, redaction, and no ANSI.

If you want, I can also produce a one-liner CLI (pnpm run log:sample) that prints a valid JSONL record (already included).

Perfect. Here’s exactly how to use the tiny JSONL sampler you now have.

Run it

```
pnpm run log:sample
```

What you’ll see (shape is fixed, timestamp will be “now”)

```
{"timestamp":"2025-10-28T13:37:42.123Z","level":"INFO","severityNumber":9,"msg":"hello","resource":{"service
.name":"orchestration-build"}}
```

Quick sanity checks (optional)

1) Parse + validate required fields

```
pnpm run log:sample | jq -e 'has("timestamp") and has("level") and has("severityNumber") and has("msg") and
(.resource["service.name"] != null)'
```

2) Ensure trailing newline (JSON Lines contract)

```
pnpm run log:sample | tail -c1 | hexdump -C
# should end with: 0a
```

3) Pipe-friendly (no ANSI)

```
pnpm run log:sample | jq .
```

All set — the command prints one clean JSON object per line to stdout, perfect for CI or manual inspection.