phase 1G and extra Logging, Provenance & Redaction (CI-friendly)

Awesome—folding your 1G research into the Phase-1 plan now. Here's a tight, implementation-ready cut you can drop in.

Phase 1 — 1G: Logging, Provenance & Redaction (CI-friendly)

Decisions (locked)

- Library: pino (JSON Lines; high-throughput).
- Format: JSONL (UTF-8, LF), one object per line, no ANSI.
- Streams: App telemetry → stderr; CLI "contract output" (diff + CONVERGED|FAILED) → stdout.
- Time/ordering: ISO-8601 UTC (TZ=UTC in CI). Use monotonic timing only for durations, not ordering.
- Trace context: If present, include W3C traceId, spanId, traceFlags via @opentelemetry/api.
- Schema fields (minimum): timestamp, level/severityNumber, msg, event, attributes{...}, traceId, spanId, traceFlags, schemaVersion.
- Redaction: Fail-closed mindset; start with strong path-based denylist in pino + allowlist discipline for what we log in attributes.
- CI hygiene: No timestamps in CLI "status line"; logs only JSONL; respect NO_COLOR. Mask any ad-hoc secrets via Actions ::add-mask:: before echo.

———

Repo changes

New files

- apps/orchestrator/src/lib/logger.ts (pino config + redaction + OTEL mixin)
- apps/orchestrator/test/logger.spec.ts (determinism + redaction tests)
- docs/logging/structured-logging.md (schema + field meanings + examples)

Deps

```
pnpm -w add pino @opentelemetry/api
pnpm -w add -D @types/pino
```

CI env

- In your workflow job: env: { TZ: UTC, NO_COLOR: '1' }

———

Code: production-ready minimal logger

```
// apps/orchestrator/src/lib/logger.ts
import pino from 'pino';
import { trace } from '@opentelemetry/api';
export type LogLevel = 'trace'|'debug'|'info'|'warn'|'error'|'fatal';
export const logger = pino({
  level: process.env.LOG_LEVEL ?? 'info',
  timestamp: pino.stdTimeFunctions.isoTime, // ISO8601 UTC
  formatters: {
    level(label) { return { level: label.toUpperCase() }; }
  },
  // Redact obvious secrets; keep this list tight and extend as we integrate providers.
  redact: {
    paths: [
      'password',
      'secret',
      'token',
      '*.apiKey',
      'req.headers.authorization',
      'req.headers.cookie',
      'user.email',
      'payment.card.number',
      'items[*].token'
    ],
    censor: '[REDACTED]',
    remove: false
  },
  // Attach trace context when available
  mixin() {
    const span = trace.getActiveSpan();
    if (!span) return {};
    const { traceId, spanId, traceFlags } = span.spanContext();
    return { traceId, spanId, traceFlags };
  },
  base: {
    service: 'orchestrator',
    schemaVersion: '1.0'
  }
}, pino.destination({ fd: 2, sync: false })); // fd 2 = stderr
export function logEvent(level: LogLevel, event: string, msg: string, attributes?: Record<string, unknown>) {
  // Enforce small, allowlisted attribute payloads
  const safe = attributes ? JSON.parse(JSON.stringify(attributes)) : undefined;
  logger[level]({ event, attributes: safe }, msg);
}
```

Tests: determinism & redaction

```
// apps/orchestrator/test/logger.spec.ts
import { describe, it, expect, vi, beforeEach, afterEach } from 'vitest';
import { Writable } from 'node:stream';
import pino from 'pino';
import { trace } from '@opentelemetry/api';
// Capture to memory (no TTY, no ANSI)
class Sink extends Writable {
  chunks: string[] = [];
  _write(chunk: any, _enc: any, cb: any) { this.chunks.push(String(chunk)); cb(); }
}
const mkLogger = () => {
  const sink = new Sink();
  const l = pino({
    timestamp: () => `,"time":"2025-01-01T00:00:00.000Z"`,
    redact: { paths: ['secret', 'req.headers.authorization'], censor: '[REDACTED]' },
    base: { service: 'test', schemaVersion: '1.0' },
    formatters: { level: (lbl: string) => ({ level: lbl.toUpperCase() }) }
  }, sink as unknown as any);
  return { l, sink };
};
describe('logger', () => {
  beforeEach(() => { process.env.NO_COLOR = '1'; });
  afterEach(() => { delete process.env.NO_COLOR; });
  it('emits JSONL with deterministic fields and no ANSI', () => {
    const { l, sink } = mkLogger();
    l.info({ event: 'unit', attributes: { a: 1 } }, 'hello');
    const line = sink.chunks[0]!.trimEnd();
    expect(() => JSON.parse(line)).not.toThrow();
    expect(line).not.toMatch(/\x1b\[/); // no ANSI
    const obj = JSON.parse(line);
    expect(obj).toMatchObject({
      level: 'INFO', msg: 'hello', event: 'unit',
      attributes: { a: 1 }, service: 'test', schemaVersion: '1.0'
    });
  });
  it('redacts sensitive fields', () => {
    const { l, sink } = mkLogger();
    l.info({ event: 'login', attributes: { secret: 'shh', req: { headers: { authorization: 'Bearer X' } } } }, 'attempt');
    const obj = JSON.parse(sink.chunks.at(-1)!);
    expect(obj.attributes.secret).toBe('[REDACTED]');
    expect(obj.attributes.req.headers.authorization).toBe('[REDACTED]');
  });
});
```

Vitest config add-ons

- Ensure TZ=UTC, watch:false, and LF EOLs.
- (Already covered in 1F; no extra changes needed.)

Doc stub (docs/logging/structured-logging.md)

- Purpose: Interop with OTEL, CI-safe JSONL.
- Field list: as above (severity mapping included).
- Do / Don’t:
- Do log: event name, bounded attributes, IDs (traceId/spanId/messageId).
- Don’t log: raw secrets/PII, huge objects, ANSI, timestamps in CLI status lines.
- Examples: ERROR & INFO samples.
- Rotation: none in Phase-1; CI logs only. (Add rotation when services persist logs.)

CLI integration (no behavior change to 1E)

- Continue printing diff + final status to stdout.
- Any diagnostic (e.g., “compiled schema”, “validator warmed”) → logEvent('info', 'cli.init', 'starting cli', {...}) → stderr.

Scripts

Add to apps/orchestrator/package.json:

```
{
  "scripts": {
    "poc:log": "node -e \"(async()=>{const {logEvent}=await import('./dist/lib/logger.js'); logEvent('info','poc.log','Logger wired',{ok:true}); console.log('CONVERGED'); process.exit(0)})()\""
  }
}
```

Note: poc:log writes structured log to stderr and CONVERGED to stdout.

Acceptance criteria (1G)

- Unit tests prove: valid JSONL, no ANSI, UTC timestamps, redaction works.
- Logger emits to stderr; CLI status remains on stdout.
- Trace fields appear when an active span exists (harmlessly absent otherwise).
- docs/logging/structured-logging.md committed with schema + examples.
- CI sets TZ=UTC and NO_COLOR=1.

If you want, I can generate a tiny PR diff for 1G (logger + tests + doc) just like we did for 1D’s mock diff—say the word and I’ll print the patch.

Here’s the tiny PR diff for 1G (logger + tests + doc) — adds three files only (no package.json edits). Apply, then install deps with:

```
pnpm -w add pino @opentelemetry/api && pnpm -w add -D @types/pino
```

```
diff --git a/apps/orchestrator/src/lib/logger.ts b/apps/orchestrator/src/lib/logger.ts
new file mode 100644
--- /dev/null
+++ b/apps/orchestrator/src/lib/logger.ts
@@ -0,0 +1,64 @@
+import pino, { type Logger } from 'pino';
+import { trace } from '@opentelemetry/api';
+
+export type LogLevel = 'trace' | 'debug' | 'info' | 'warn' | 'error' | 'fatal';
+
+export const logger: Logger = pino({
+  level: process.env.LOG_LEVEL ?? 'info',
+  timestamp: pino.stdTimeFunctions.isoTime, // ISO-8601 UTC
+  formatters: {
+    level(label: string) {
+      return { level: label.toUpperCase() };
+    }
+  },
+  // Conservative default redactions; extend per provider as needed.
+  redact: {
+    paths: [
+      'password',
+      'secret',
+      'token',
+      '*.apiKey',
+      'req.headers.authorization',
+      'req.headers.cookie',
+      'user.email',
+      'payment.card.number',
+      'items[*].token'
+    ],
+    censor: '[REDACTED]',
+    remove: false
+  },
+  // Attach W3C trace context when an active span exists.
+  mixin() {
+    const span = trace.getActiveSpan();
+    if (!span) return {};
+    const { traceId, spanId, traceFlags } = span.spanContext();
+    return { traceId, spanId, traceFlags };
+  },
+  base: {
+    service: 'orchestrator',
+    schemaVersion: '1.0'
+  }
+  // Destination is stderr by default in Node if not overridden; explicit fd=2 below.
+}, pino.destination({ fd: 2, sync: false }));
+
+export function logEvent(
+  level: LogLevel,
+  event: string,
+  msg: string,
+  attributes?: Record<string, unknown>
+) {
+  // Shallow JSON round-trip to drop non-serializable values deterministically.
+  const safe = attributes ? JSON.parse(JSON.stringify(attributes)) : undefined;
+  (logger as any)[level]({ event, attributes: safe }, msg);
+}
+
+export default logger;
diff --git a/apps/orchestrator/test/logger.spec.ts b/apps/orchestrator/test/logger.spec.ts
new file mode 100644
--- /dev/null
+++ b/apps/orchestrator/test/logger.spec.ts
@@ -0,0 +1,74 @@
+import { describe, it, expect } from 'vitest';
+import { Writable } from 'node:stream';
+import pino from 'pino';
+
+class Sink extends Writable {
+  chunks: string[] = [];
+  _write(chunk: any, _enc: any, cb: any) {
+    this.chunks.push(String(chunk));
+    cb();
+  }
+}
+
```

```diff
+const mkLogger = () => {
+  const sink = new Sink();
+  const l = pino({
+    // Deterministic timestamp for snapshot-like assertions.
+    timestamp: () => `,"time":"2025-01-01T00:00:00.000Z"`,
+    redact: {
+      paths: ['secret', 'req.headers.authorization'],
+      censor: '[REDACTED]'
+    },
+    base: { service: 'test', schemaVersion: '1.0' },
+    formatters: { level: (lbl: string) => ({ level: lbl.toUpperCase() }) }
+  }, sink as unknown as any);
+  return { l, sink };
+};
+
+describe('logger', () => {
+  it('emits JSONL without ANSI and with expected fields', () => {
+    const { l, sink } = mkLogger();
+    l.info({ event: 'unit', attributes: { a: 1 } }, 'hello');
+    const line = sink.chunks[0]!.trimEnd();
+    expect(() => JSON.parse(line)).not.toThrow();
+    expect(line).not.toMatch(/\x1b\[/); // no ANSI codes
+    const obj = JSON.parse(line);
+    expect(obj).toMatchObject({
+      level: 'INFO',
+      msg: 'hello',
+      event: 'unit',
+      attributes: { a: 1 },
+      service: 'test',
+      schemaVersion: '1.0'
+    });
+  });
+
+  it('redacts sensitive fields', () => {
+    const { l, sink } = mkLogger();
+    l.info(
+      { event: 'login', attributes: { secret: 'shh', req: { headers: { authorization: 'Bearer X' } } } },
+      'attempt'
+    );
+    const obj = JSON.parse(sink.chunks.at(-1)!);
+    expect(obj.attributes.secret).toBe('[REDACTED]');
+    expect(obj.attributes.req.headers.authorization).toBe('[REDACTED]');
+  });
+});
diff --git a/docs/logging/structured-logging.md b/docs/logging/structured-logging.md
new file mode 100644
--- /dev/null
+++ b/docs/logging/structured-logging.md
@@ -0,0 +1,44 @@
+# Structured Logging (JSONL) — Phase 1G
+
+**Stream & format**
+- Emit **JSON Lines** to **stderr** (one JSON object per line, UTF-8, LF).
+- No ANSI escape codes. CLI contract output (diff + `CONVERGED|FAILED`) stays on **stdout**.
+- Timestamps are ISO-8601 UTC; set `TZ=UTC` in CI.
+
+**Fields (minimum)**
+- `level` (INFO/WARN/ERROR), `msg` (string), `event` (string), `attributes` (object)
+- `traceId`, `spanId`, `traceFlags` (present when OTEL context is active)
+- `service`, `schemaVersion`
+
+**Redaction**
+- Default redactions: `password`, `secret`, `token`, `*.apiKey`,
+  `req.headers.authorization`, `req.headers.cookie`, `user.email`,
+  `payment.card.number`, `items[*].token`.
+- Extend per integration; prefer allowlisting for high-risk contexts.
+
+**Example line**
+```json
+{"level":"INFO","time":"2025-01-01T00:00:00.000Z","event":"unit","attributes":{"a":1},"msg":"hello","service":"orchestrator","schemaVersion":"1.0"}
+```
+
+**Install**
+```bash
+pnpm -w add pino @opentelemetry/api
+pnpm -w add -D @types/pino
+```
+
+**Notes**
+- Do not rely on log ordering in CI; use timestamps and/or `traceId` correlation.
+- Keep `attributes` bounded and serializable; avoid large/binary payloads.
```

Got it—folding your 1G research into the plan and giving you a tiny, PoC-friendly PR diff.

1G — Static Analysis & Type Safety (Fold-in)

Decisions (PoC speed > heaviness):

- TypeScript: strict + safety opts: noUncheckedIndexedAccess, exactOptionalPropertyTypes, noImplicitOverride, useUnknownInCatchVariables, noFallthroughCasesInSwitch; ESM via module/moduleResolution: "NodeNext"; skipLibCheck: true; CI uses tsc --noEmit.
- ESLint (Flat): @eslint/js + @typescript-eslint recommended (non-type-checked) to keep it fast; add eslint-config-prettier last to kill formatting conflicts; ignore dist/.
- Prettier: run separately; minimal .prettierrc; no eslint-plugin-prettier.
- Node: 22 LTS (engines + CI image).

Tiny PR diff (adds TS/ESLint/Prettier + fast CI)

```diff
--- a/package.json
+++ b/package.json
@@ -1,6 +1,23 @@
 {
   "name": "orchestration-build",
   "version": "0.0.0",
+  "engines": { "node": ">=22" },
   "private": true,
   "scripts": {
-    "test": "echo \"no tests yet\""
+    "test": "echo \"no tests yet\"",
+    "type-check": "tsc --noEmit",
+    "lint": "eslint .",
+    "lint:fix": "eslint --fix .",
+    "format": "prettier --write .",
+    "check:fast": "pnpm format && pnpm lint && pnpm type-check"
   },
+  "devDependencies": {
+    "@eslint/js": "^9.0.0",
+    "eslint": "^9.0.0",
+    "eslint-config-prettier": "^9.1.0",
+    "prettier": "^3.3.3",
+    "typescript": "^5.6.0",
+    "typescript-eslint": "^8.0.0"
```

```
+  }
 }
--- /dev/null
+++ b/tsconfig.json
@@ -0,0 +1,34 @@
+{
+  "compilerOptions": {
+    "target": "ES2022",
+    "module": "NodeNext",
+    "moduleResolution": "NodeNext",
+    "strict": true,
+    "noUncheckedIndexedAccess": true,
+    "exactOptionalPropertyTypes": true,
+    "noImplicitOverride": true,
+    "useUnknownInCatchVariables": true,
+    "noFallthroughCasesInSwitch": true,
+    "skipLibCheck": true,
+    "esModuleInterop": true,
+    "forceConsistentCasingInFileNames": true,
+    "outDir": "./dist",
+    "rootDir": "./src"
+  },
+  "include": ["src/**/*.ts", "eslint.config.mjs"],
+  "exclude": ["node_modules", "dist"]
+}
--- /dev/null
+++ b/eslint.config.mjs
@@ -0,0 +1,24 @@
+// @ts-check
+import eslint from '@eslint/js';
+import tseslint from 'typescript-eslint';
+import eslintConfigPrettier from 'eslint-config-prettier';
+
+export default tseslint.config(
+  { ignores: ['dist/', 'node_modules/'] },
+  eslint.configs.recommended,
+  tseslint.configs.recommended,
+  // Keep it fast for PoC; upgrade to recommendedTypeChecked later.
+  eslintConfigPrettier // MUST be last to disable formatting conflicts
+);
--- /dev/null
+++ b/.prettierrc
@@ -0,0 +1,8 @@
+{
+  "semi": true,
+  "singleQuote": true,
+  "trailingComma": "es5",
+  "printWidth": 100,
+  "arrowParens": "always"
+}
--- /dev/null
+++ b/.github/workflows/checks.yml
@@ -0,0 +1,41 @@
+name: fast-checks
+on:
+  pull_request:
+  push:
+    branches: [ main ]
+jobs:
```

````
+  lint-and-types:
+    runs-on: ubuntu-latest
+    steps:
+      - uses: actions/checkout@v4
+      - uses: pnpm/action-setup@v4
+        with:
+          version: 9
+      - uses: actions/setup-node@v4
+        with:
+          node-version: '22.x'
+          cache: 'pnpm'
+      - name: Install
+        run: pnpm i --frozen-lockfile
+      - name: Lint (no formatting conflicts)
+        run: pnpm lint --max-warnings=0
+      - name: Type-check (noEmit)
+        run: pnpm type-check
--- /dev/null
+++ b/docs/quality.md
@@ -0,0 +1,31 @@
+# Static Analysis & Type Safety (PoC)
+
+**Why this setup?** High-signal checks without slowing the loop.
+
+## TypeScript
+- `strict` +: `noUncheckedIndexedAccess`, `exactOptionalPropertyTypes`,
+  `noImplicitOverride`, `useUnknownInCatchVariables`, `noFallthroughCasesInSwitch`.
+- ESM via `module`/`moduleResolution: NodeNext`. CI: `tsc --noEmit`.
+
+## ESLint (Flat)
+- `@eslint/js` + `typescript-eslint` **recommended** (non type-checked).
+- `eslint-config-prettier` last to prevent rule conflicts.
+
+## Prettier
+- Run separately from ESLint; `.prettierrc` defines formatting.
+
+## Node
+- Require Node **22 LTS** locally and in CI.
+
+## Commands
+```sh
+pnpm format
+pnpm lint
+pnpm lint:fix
+pnpm type-check
+pnpm check:fast
+```
````

If you want, I can also generate a tiny “red bar” sample (one deliberate lint + one TS error) in src/example.ts to prove the guards trip, but I kept the diff lean per PoC.

Perfect—here’s a tiny “red bar” file that triggers:

- ESLint (unused var), and
- TypeScript (strict + noUncheckedIndexedAccess → possibly-undefined arg)

```
--- /dev/null
+++ b/src/example.ts
@@ -0,0 +1,20 @@
+/**
+ * Intentional red-bar sample for CI:
+ * - ESLint: unused variable 'unused'
```

```
+ * - TypeScript: 'noUncheckedIndexedAccess' makes arr[0] => string | undefined,
+ *   causing an error when passed where string is required.
+ */
+
+// LINT violation (unused):
+const unused = 42;
+
+export function shout(s: string): string {
+  return s.toUpperCase();
+}
+
+export function demo(arr: string[]): string {
+  // With "noUncheckedIndexedAccess": true, type of arr[0] is string | undefined
+  const first = arr[0]; // first: string | undefined
+  // TS error: Argument of type 'string | undefined' is not assignable to parameter of type 'string'.
+  return shout(first);
+}
```

Run to see failures:

- `pnpm lint --max-warnings=0` → unused var trips ESLint
- `pnpm type-check` → TS error on shout(first)