SOFTWARE AUDITABLE

Fundamentos Teóricos, Técnicos y Regulatorios



Diego Javier Trejo España
Iván Danilo García Santillán

SOFTWARE AUDITABLE

Fundamentos Teóricos, Técnicos y Regulatorios

Diego Javier Trejo España

Iván Danilo García Santillán

SOFTWARE AUDITABLE

Fundamentos Teóricos, Técnicos y Regulatorios

Diego Javier Trejo España

Iván Danilo García Santillán

UNIVERSIDAD TÉCNICA DEL NORTE
UTN
IBARRA - ECUADOR

EDITORIAL UTN
IBARRA - ECUADOR

Edita

Editorial Universidad Técnica del Norte
Av. 17 de Julio, 5-21. Campus los Olivos
Ibarra – Imbabura - República del Ecuador
Telf. 593 (6) 299 7800
URL: www.utn.edu.ec
Editorial: editorial@utn.edu.ec

Pares revisores académicos externos

Mgs. Darwin Pillo Guanoluisa
dmpillo@pucesi.edu.ec
PUCE - Ibarra
Mgs. Segundo Pusdá Chulde
sepusda@pucesi.edu.ec
PUCE - Ibarra

Revisión de estilo

Mgs. Silvia Arciniega Hidrobo
srarciniega@utn.edu.ec
UTN - Ibarra

Cómo citar la obra completa

Trejo-España, D. y García-Santillán, I. (2026). Software auditable: Fundamentos teóricos, técnicos y regulatorios. Editorial Universidad Técnica del Norte (UTN).

Dirección de arte y diagramación

Pedro David Cargua Gualoto
pdcarguag@utn.edu.ec
Universidad Técnica del Norte, Ecuador

Número de páginas

145

© de los textos y fotografías: Sus respectivos autores, 2026.
© de esta edición: Editorial Universidad Técnica del Norte, 2026.
1ª edición, digital: junio de 2024 / e-ISBN:978-9942-572-59-2
DOI: 10.53358/libfica/NYDE4146
ROR: https://ror.org/03f0t8b71
Impreso en Ecuador / Printed in Ecuador
Prohibida la reproducción total o parcial de esta obra sin la previa autorización escrita de la Editorial Universidad Técnica del Norte.

Autores

Diego Javier Trejo España
Universidad Técnica del Norte, Ecuador
https://orcid.org/0000-0002-2973-4345
djtrejo@utn.edu.ec

Iván Danilo García Santillán
Universidad Técnica del Norte, Ecuador
https://orcid.org/0000-0001-6404-5185
idgarcia@utn.edu.ec

AUTORES

DIEGO JAVIER TREJO ESPAÑA

Magíster en Gerencia Informática – Pontificia Universidad Católica del Ecuador.
Ingeniero en Sistemas Computacionales.

Docente a Tiempo Completo – Carrera de Ingeniería en Sistemas Computacionales, UTN.
Consultor en tecnología y gestión de proyectos informáticos.

djtrejo@utn.edu.ec
https://orcid.org/0000-0002-2973-4345

IVÁN DANILO GARCÍA SANTILLÁN

Ingeniero en Sistemas Informáticos.
Ph.D. en Ingeniería Informática – Universidad Complutense de Madrid, España.
Maestría en Gerencia Informática.
Diploma en Docencia Universitaria.

Profesor asociado – Carrera de Sistemas Informáticos, UTN.
Investigador acreditado por SENESCYT.

Experiencia en docencia universitaria, desarrollo de sistemas y dirección de proyectos de investigación en Ingeniería de Software e Inteligencia Artificial.
Publicaciones científicas en revistas de alto impacto y participación en congresos internacionales.

Áreas de interés: Inteligencia Artificial, Minería de Datos, Big Data, Aprendizaje Automático, Aprendizaje Profundo y Ciencia de Datos.

idgarcia@utn.edu.ec
https://orcid.org/0000-0001-6404-5185

Nota sobre herramientas de asistencia

La redacción de este documento se realizó con asistencia de herramientas de inteligencia artificial para optimización de estilo y estructura de texto. Todo el contenido técnico, investigación de fuentes, verificación de datos, y argumentación son propios de los autores. Las referencias bibliográficas fueron consultadas y verificadas personalmente.

Resumen

Este libro ofrece una visión general del concepto de software auditable, desde los antecedentes teóricos, el diseño técnico, los contextos legales y normativos hasta su implementación práctica en el entorno de producción. La revisión bibliográfica incorpora resultados conceptualizaciones más recientes en el ámbito de la inteligencia artificial y las aplicaciones distribuidas nativas de la nube.

En los últimos años, el software auditable ha pasado de ser un requisito deseable a una necesidad para aplicaciones de alto impacto: por un lado, debido al creciente número de escándalos corporativos masivos; y por otro, debido a la reciente promulgación de leyes de protección de datos a nivel internacional y nacional. En este sentido, según el modelo de Sieber y Partners, los tres círculos de auditabilidad se definen como: legibilidad del código, comprensibilidad arquitectónica y documentación exhaustiva, que se analizan en este libro.

Además, se analiza el contexto normativo internacional, basado en la norma IEEE 1028-2008 y el modelo de calidad ISO/IEC 25010. Así como las limitaciones técnicas introducidas por marcos legales como el GDPR, la HIPAA y la Ley Sarbanes-Oxley. Finalmente, se proporciona un manual de implementación práctica para el marco de desarrollo .NET, donde se aplican los conceptos teóricos en arquitecturas concretas mediante: la biblioteca Serilog para el registro estructurado; Entity Framework Core para la auditoría a nivel de datos; y patrones de diseño para la trazabilidad integral.

Índice general

Nota sobre herramientas de asistencia 1
Resumen 2

Capítulo 1
INTRODUCCIÓN Y CONTEXTO HISTÓRICO 11
1.1 La Era Digital y la Crisis de Confianza Computacional 12
1.2 Escándalos que motivaron al software auditable . . . 14
1.3 El Caso Enron: Fraude Facilitado por Tecnología . . 14
1.4 Escándalos en América Latina: Contexto Regional . . 16
1.5 Odebrecht (2014-2016) 16
1.6 Petrobras (Brasil, 2014) 16
1.7 Crisis Ecuatoriana (1990-2020) 17
1.7.1 Crisis bancarias 17
1.7.2 Petroecuador y Contratos con Odebrecht (2007-2016) 18
1.7.3 Sobreprecio en Compras Públicas COVID-19 (2020) . 19
1.8 Lecciones para Sistemas Auditables 20
1.9 La Respuesta Legislativa: Sarbanes-Oxley Act de 2002 21
1.10 Estructura y Provisiones Clave de SOX 23
1.11 La Respuesta Legislativa Ecuatoriana 23
1.11.1 Ley Orgánica de Empresas Públicas 24
1.11.2 Contraloría General del Estado: NCI 25
1.11.3 La Norma 500: Información y Comunicación 26
1.11.4 Código Orgánico Monetario y Financiero (2014) . . . 26
1.12 El Caso por el Software Auditable: Evidencia Empírica 27

Capítulo 2
FUNDAMENTOS CONCEPTUALES **29**
2.1 Definición Operacional de Auditabilidad y Software Auditable 32
2.2 Definición y Marco Conceptual del Registros de Auditoría 33
2.3 Anatomía de un Registro de Auditoría (Audit Trail) . 35
2.3.1 Sistemas de Logging Estructurado 35
2.3.2 Quién (Who) 35
2.3.3 Qué (What) 35
2.3.4 Cuando (When) 36
2.3.5 Dónde (Where) 36
2.3.6 Cómo (How) 36
2.3.7 Por qué (Why) 37
2.4 Requisitos de Integridad y No Repudio 38
2.5 Registros de Auditoría en Bases de Datos 41
2.5.1 Auditoría basada en triggers 42
2.5.2 Auditoría mediante transaction logs 42
2.5.3 Auditoría a nivel de aplicación 43

Capítulo 3
MARCOS REGULATORIOS Y NORMATIVOS **44**
3.1 El Estándar IEEE 1028-2008 45
3.2 ISO/IEC 25010: Modelo de Calidad de Software . . . 45
3.3 GDPR: El Reglamento General de Protección de Datos 46
3.4 HIPAA y los Controles de Auditoría 47
3.5 Principios Fundamentales y el Concepto de Accountability 47
3.6 Protección de Información de Salud en Estados Unidos 48
3.7 Protección de Datos Personales (GDPR) en Ecuador 50
3.7.1 Influencia del GDPR europeo en Ecuador 50
3.7.2 Requisitos de Auditoría y Accountability del GDPR . 51
3.7.3 Marco Constitucional Ecuatoriano de Protección de Datos 53
3.7.4 Ley Orgánica de Protección de Datos Personales (2021) 53
3.7.5 Reglamento de la LOPDP (2023) 56

3.8 Implicaciones para Sistemas Auditables en Ecuador . 57

Capítulo 4
ARQUITECTURA DE SOFTWARE AUDITABLE 60
4.1 Modelo de Cinco Capas para Sistemas Auditables . . 61
4.2 Taxonomía de Eventos Auditables 61
4.3 Clasificación Primaria: Eventos Técnicos versus Eventos de Negocio 62
4.3.1 Eventos Técnicos 62
4.3.2 Eventos de Negocio 65
4.4 Principios Fundamentales de Diseño 65
4.5 Segregación de Responsabilidades de Auditoría . . . 66
4.6 Inmutabilidad de Registros de Auditoría 67
4.7 Paradigma Append-Only para Registros de Auditoría 67
4.8 Principio de Mínimo Privilegio para Acceso a Auditoría 68
4.9 Captura y enriquecimiento de eventos 68
4.10 Almacenamiento Seguro de Registros de Auditoría . . 69
4.10.1 Captura de Contexto Enriquecido 69
4.10.2 Contexto de Identidad y Autenticación 69
4.11 Append-Only Databases 70
4.11.1 Concepto y Principios Fundamentales 70
4.11.2 Inmutabilidad y Temporal Semantics 71
4.11.3 Log-Structured Storage Engines 72
4.11.4 Event Stores Especializados 74
4.11.5 Implementación en Bases de Datos Relacionales . . . 75
4.11.6 Trade-offs y consideraciones de rendimiento 76
4.12 Desafíos y Limitaciones 77
4.13 Casos de Uso Óptimos 79
4.14 Implementación de Trazas de Auditoría 80
4.15 Protección Criptográfica de Registros de Auditoría . 81
4.15.1 Hash Chains para Detección de Manipulación 81
4.15.2 Merkle Trees para Verificación Eficiente 82
4.15.3 Firmas Digitales y No Repudio 83
4.15.4 Timestamping Confiable (RFC 3161) 84
4.16 Análisis forense de registros de auditoría 85
4.16.1 Metodología de Investigación Forense Digital 85

4.16.2 Reconstrucción de Líneas de Tiempo de Eventos . . . 85
4.16.3 Detección de Alteración de Logs 86

Capítulo 5
ANÁLISIS Y GENERACIÓN DE REPORTES DE AUDITORÍA 87
5.1 Análisis de Datos de Auditoría 88
5.1.1 Etapa 1: Recolección y Agregación 88
5.2 Etapa 2: Normalización y Análisis Sintáctico (Parsing) 89
5.3 Etapa 3: Enriquecimiento 89
5.4 Etapa 4: Filtrado y Reducción 91
5.5 Etapa 5: Correlación y Agregación 92
5.6 Técnicas de Análisis: De Descriptivo a Predictivo . . 93
5.6.1 Análisis Descriptivo: ¿Qué Ocurrió? 93
5.6.2 Análisis Diagnóstico: ¿Por Qué Ocurrió? 94
5.6.3 Análisis Predictivo: ¿Qué Ocurrirá? 95
5.7 Plataformas Integradas 97

Capítulo 4
CASOS DE USO VERTICALES 99
6.1 Sector Financiero y Fintech 100
6.2 Sector Salud: Registros Médicos Electrónicos 101
6.3 Facturación Electrónica en Ecuador: Sistema SRI . . 101
6.3.1 Arquitectura del Sistema de Facturación Electrónica 102
6.3.2 Implementación de Firma Digital XAdES-BES 102
6.3.3 Proceso de Autorización y Rastro de Auditoría 103
6.4 BANRED Ecuador: Switch Interbancario y Compliance PCI-DSS 103

Capítulo 7
IMPLEMENTACIÓN PRÁCTICA EN ENTORNOS .NET 106
7.1 Diseño de Arquitectura para un Software Auditable . 107
7.2 Guía de Implementación de Serilog en .NET Core . . 108
7.2.1 Fundamentos de Configuración mediante appsettings.json 109
7.2.2 Instalación de Paquetes Necesarios 109

7.2.3 Integración en Program.cs 110
7.2.4 Estructura de Configuración en appsettings.json . . . 111
7.2.5 Configuración Específica por Ambiente 114
7.2.6 Formato Estructurado mediante JSON Compacto . . 115
7.2.7 Recarga Dinámica de Configuración 116
7.2.8 Uso de Serilog según Nivel de Verbosidad 117
7.2.9 Ejemplo de Uso de Serilog en un proyecto .NET Core 123

Capítulo 8
CONCLUSIONES **133**

Índice de figuras

2 **Figura 2:** *Trazabilidad de una transacción de extremo a extremo en un sistema auditable* 31
3 **Figura 3:** *Mecanismo de firma digital para garantía de no repudio en registros de auditoría* 39
4 **Figura 4:** *Cadena de hashes criptográficos para integridad de registros* 40
5 **Figura5:** *Arquitectura de una aplicación .Net Core usando Serilog* 124
6 **Figura 6:** *Pipeline de flujo de Serilog* 126
7 **Figura 7:** *Componentes fundamentales para la implementación de Serilog en aplicaciones .NET* 127

Índice de tablas

1 **Tabla 1:** *Comparación SOX vs. Marco Ecuatoriano* . 27
2 **Tabla 2:** *Pilares Fundamentales del Software Auditable* 33
3 **Tabla 3:** *Comparación GDPR vs. LOPDP* 59
4 **Tabla 4:** *Capas de los Sistemas Auditables* 61

Capítulo 1

INTRODUCCIÓN Y CONTEXTO HISTÓRICO

1.1 La Era Digital y la Crisis de Confianza Computacional

En las primeras dos décadas del siglo XXI, el software ha penetrado prácticamente todos los aspectos de la actividad humana organizada, convirtiéndose en la infraestructura invisible pero omnipresente sobre la cual se construye la sociedad contemporánea. Hoy en día, los sistemas de información manejan más de 5 billones de dólares en transacciones por día en los mercados financieros globales, procesan información de salud de más de 7 mil millones de personas en sistemas de salud digitales, controlan gran parte de la infraestructura crítica que proporciona nuestra electricidad, agua, transporte y otros servicios vitales, y facilitan gran parte de la interacción y el comercio entre miles de millones de personas que utilizan plataformas digitales que operan a escala planetaria (Bank for International Settlements, 2025).

Esta dependencia ubicua ha generado lo que académicos llaman una "crisis de confianza computacional" (MacKenzie, 2001). A diferencia de sistemas físicos cuyos comportamientos pueden ser verificados mediante inspección directa o pruebas empíricas repetibles, los sistemas de software operan frecuentemente como 'cajas negras' cuyo funcionamiento interno permanece opaco incluso para expertos técnicos altamente capacitados. Esta falta de transparencia plantea riesgos sistémicos cuando software defectuoso o malicioso se ejecuta en entornos donde los fallos podrían tener efectos devastadores en personas, empresas o incluso sociedades enteras.

Cabe destacar el concepto de "caja negra" en los sistemas informáticos. Dijstra (Dijkstra, s.f.) y Mccloure (McClure, 2001) escriben: "Los numerosos elementos que componen un sistema tecnológico son fijos, en el sentido de que no forman parte de una negociación social ni darán lugar a nuevas negociaciones sociales. Un buen programa o un cuerpo inquebrantablemente rígido pueden utilizarse, en su mayor parte, como cajas negras; en general, una vez que la máquina funciona, ya nadie tiene que preocuparse por ella". En el ámbito del software moderno, existen cajas negras en cada capa: el código fuente se compila en binarios inescrutables, las aplicaciones de

microservicios implican que una sola transacción comercial implica llamar a docenas de servicios independientes, y el entrenamiento de modelos de aprendizaje automático permite que los modelos descubran patrones que sus propios creadores no pueden explicar.

El costo de la falta de auditabilidad tiene consecuencias directas y fácilmente medibles. Según el estudio del Consorcio para la Información y la Calidad del Software (CISQ) de 2022, tan solo el problema de la calidad del software le cuesta a la economía estadounidense alrededor de 2,41 trillones de dólares al año. Estos no son solo los costes directos de las fallas de software (tiempo de inactividad del sistema, pérdida de datos transaccionales, recuperación ante desastres), sino también los costes indirectos (pérdida de productividad, pérdida de reputación, oportunidades de negocio perdidas) (Krasner, 2022).

Aún más alarmante, el informe anual sobre filtraciones de datos de IBM de 2023 (Fantl, n.d.) ofrece una idea de las consecuencias de no contar con sistemas auditables. El coste medio de una filtración de datos ha ascendido a 4,45 millones de dólares, un incremento del 15 % desde 2020. Pero el dato más revelador del informe no es el coste de la filtración en sí, sino la distribución de las causas raíz. Dos tercios de los incidentes de seguridad fueron causados por acciones involuntarias de usuarios internos autorizados; no por hackers externos maliciosos, sino por empleados autorizados que cometieron un error o hicieron clic en un correo electrónico de phishing. Aún más preocupante, el 88 % de todas las brechas de seguridad fueron causadas por errores humanos en algún punto de la cadena causal.

Estos datos muestran claramente que el problema no radica en la mayor sofisticación de los atacantes, sino en la imposibilidad de que las organizaciones monitoreen, comprendan o controlen lo que hacen los usuarios autorizados dentro de los sistemas. Sin registros de auditoría adecuados ni detección de anomalías, las organizaciones operan a ciegas, asumiendo que decenas de miles, o decenas de millones de accesos a información confidencial se realizan correctamente, sin ninguna forma de probar o siquiera verificar si esa suposición es correcta.

1.2 Escándalos que motivaron al software auditable

La necesidad de software auditable no surgió del vacío académico, sino que se cristalizó dramáticamente a finales del siglo XX con una serie de colapsos corporativos espectaculares que revelaron fallas sistémicas en los sistemas de información financiera y en los procesos de auditoría externa. Estos escándalos no solo destruyeron compañías individuales, sino que sacudieron la confianza pública en los mercados de capital y forzaron una reevaluación fundamental de cómo se auditan y regulan las organizaciones.

1.3 El Caso Enron: Fraude Facilitado por Tecnología

Enron Corporation representa el caso paradigmático de fraude corporativo facilitado por sistemas de información inadecuadamente auditables. Fundada en 1985 por Kenneth Lay mediante la fusión de Houston Natural Gas e InterNorth, Enron se transformó durante la década de 1990 de una compañía tradicional de gas natural en un conglomerado de trading energético que operaba en múltiples mercados globales. Avanzando rápidamente hasta el año 2000: Enron era la séptima empresa más grande de Estados Unidos por capitalización bursátil, con 111 000 millones de dólares en ingresos y alrededor de 20,000 empleados (Bondarenko, 2026).

Se consideraba una empresa innovadora y se estudiaba a menudo en las clases de negocios como ejemplo de cómo modernizar las operaciones corporativas. Durante seis años consecutivos, entre 1996 y 2001, fue nombrada “La empresa más innovadora de Estados Unidos” por Fortune. El precio de sus acciones alcanzó un máximo de 90,75 dólares a mediados de agosto de 2000. Sin embargo, todo esto era un castillo de naipes, ya que su éxito financiero se basaba en fraudes contables, facilitados por sistemas de información diseñados para eludir los controles normales (Levin Center for Oversight and Democracy, n.d.).

Enron utilizó diversas técnicas para cometer su fraude, una de las cuales se denominaba “contabilidad a valor de mercado”. Este proceso consistía en valorar los contratos a largo plazo basándose en

el valor actual neto del flujo de pagos futuros esperados. Esta técnica está permitida por los Principios de Contabilidad Generalmente Aceptados (PCGA), pero en el caso de Enron se implementó de forma fraudulenta. Enron utilizó previsiones de ingresos excesivamente optimistas para inflar el valor de los contratos y luego reconocer esos ingresos inmediatamente, en lugar de distribuirlos a lo largo de la vigencia del contrato.

Una segunda técnica fue el uso de "Entidades de Propósito Especial" (EPE), esencialmente sociedades fantasmas, creadas por Enron para comprar parte de su deuda o inversiones de bajo rendimiento. En total, Enron creó alrededor de 3.000 EPE, algunas de ellas con sede en un paraíso fiscal del Caribe. Enron utilizó las EPE para ocultar su deuda de su balance general, así como para vender inversiones de bajo rendimiento y a cometer el fraude contable. Finalmente, el director financiero de Enron, Andrew Fastow, controlaba varias de las SPE y las utilizaba para beneficiarse personalmente de las transacciones fraudulentas de Enron (Bondarenko, 2026).

En este caso, los sistemas de información de Enron estaban diseñados para manipular las transacciones que involucraban a las SPE, de modo que no quedara claro que en realidad estaban tratando consigo mismas. Los sistemas de informes estaban diseñados para que la información generada cumpliera con la letra de las normas contables, pero no reflejara el espíritu de las mismas. Los controles internos eran débiles o inexistentes, lo que permitió que se produjera el fraude.

Al final, el castillo de naipes estaba destinado a derrumbarse, y así fue. En octubre de 2001, Enron reportó una pérdida de $638 millones en ganancias netas y una disminución de $1.2 mil millones en el capital social debido a transacciones relacionadas con SPE. La SEC respondió casi de inmediato con una investigación. El precio de las acciones se desplomó casi de inmediato, pasando de $90.75 a menos de $1 en cuestión de semanas. Enron se declaró en bancarrota el 2 de diciembre de 2001, la mayor bancarrota en la historia de Estados Unidos hasta ese momento. Los accionistas perdieron aproximadamente $74 mil millones en valor de mercado (Levin Center for Oversight and

Democracy, n.d.).

1.4 Escándalos en América Latina: Contexto Regional

América Latina ha experimentado múltiples escándalos de corrupción, fraude financiero y malversación de fondos públicos que han erosionado significativamente la confianza ciudadana en instituciones públicas y privadas. Estos escándalos han expuesto debilidades sistémicas en controles internos, supervisión regulatoria, y transparencia de sistemas de información, subrayando la necesidad crítica de sistemas auditables en sectores público y privado. A continuación, se citan unos casos emblemáticos regionales.

1.5 Odebrecht (2014-2016)

El escándalo de corrupción más extenso en la historia de América Latina involucró a la constructora brasileña Odebrecht (EFE, 2025), que admitió pagar más de $788 millones en sobornos a funcionarios en 12 países latinoamericanos para asegurar contratos de infraestructura. La investigación reveló que Odebrecht operaba un "Departamento de Operaciones Estructuradas" que gestionaba pagos ilícitos mediante sistemas offshore complejos. La ausencia de registros de auditoría adecuados en sistemas gubernamentales de contratación facilitó que estos pagos permanecieran ocultos por años. El caso resultó en investigaciones en Perú, Ecuador, Colombia, México, Panamá, República Dominicana, Venezuela, Argentina, y otros países, derribando gobiernos y encarcelando presidentes y ministros (Galero, 2024).

1.6 Petrobras (Brasil, 2014)

La operación "Lava Jato" (Guerrero, 2019) descubrió un esquema masivo de corrupción en Petrobras, la petrolera estatal brasileña, donde ejecutivos aceptaban sobornos de contratistas a cambio de contratos inflados. Se estima que el esquema defraudó más de $2 mil millones. La investigación expuso que los sistemas de auditoría interna de Petrobras habían sido comprometidos deliberadamente, con auditores cómplices o coaccionados para no reportar irregulari-

dades. La falta de logs inmutables de aprobaciones de contratos y transacciones financieras dificultó rastrear el flujo completo de fondos (Guerrero, 2019).

1.7 Crisis Ecuatoriana (1990-2020)

1.7.1 Crisis bancarias

En los años 1994 al 2000, Ecuador sufrió una grave crisis bancaria provocada por la Ley General de Instituciones Financieras de 1994, que permitió los llamados "créditos vinculados", préstamos que los bancos otorgaban a sus propios accionistas y empresas del grupo sin garantías reales. El caso más emblemático fue Filanbanco, que quebró el 2 de diciembre de 1998 y requirió un salvataje estatal de $849 millones (70 % de la reserva monetaria del país). La investigación reveló que Filanbanco otorgaba préstamos sin interés a sus propios accionistas mientras cobraba anatocismo (interés sobre interés) a clientes regulares. La ausencia de sistemas para registrar adecuadamente los conflictos de interés y las aprobaciones de créditos facilitó estas prácticas. La crisis resultó en el cierre de 18 instituciones financieras y un costo de salvataje entre $5.000-6.000 millones, equivalente aproximadamente al 8 % del PIB (Carrillo Mañay et al., 2019).

Al investigar, salieron a la luz varias causas, como fallos en los sistemas de información y auditoría que permitieron préstamos a parientes sin chequear nada, estados financieros manipulados y hasta fuga de capitales. Eso destruyó la confianza en todo el sistema, también trajo cambios grandes en las reglas, por ejemplo, haciendo más fuerte a la Superintendencia de Bancos y pidiendo más detalles en la información y las auditorías.

En el caso de Filanbanco, se evidenció que usaban dos contabilidades al mismo tiempo, una oficial para los reguladores y otra real que solo unos pocos conocían. Sin controles de acceso o registros que no se pudieran cambiar, las operaciones raras no tenían un rastro claro de quién aprobaba qué, lo que facilitó todo el fraude (Carrillo Mañay et al., 2019).

Las consecuencias macroeconómicas de esta ausencia de controles

fueron devastadoras y cuantificables. Según el Banco Central del Ecuador, la crisis financiera de finales de los noventa generó pérdidas estimadas en USD 6.170 millones para el país (Telégrafo, 2014). En 1999 cerraron 18 entidades financieras, entre ellas Financorp, Azuay, Finagro, Banco del Progreso, Bancomex, Banco Popular, Banco Unión, Banco de Crédito y Filanbanco (Erazo, 2022). El salvataje estatal de Filanbanco, para entonces el banco más grande del país, demandó USD 849 millones, equivalentes al 70 % de la reserva monetaria del Ecuador en ese momento (Erazo, 2022). Adicionalmente, según el informe de Deloitte & Touche de 2001, aprobado por la Junta Bancaria, las pérdidas propias de Filanbanco ascendieron a USD 661,5 millones (Perdomo & Mogollón, 2019). Estos números ilustran con precisión qué ocurre cuando los sistemas de información financiera carecen de registros inmutables, controles de acceso auditables y flujos de aprobación documentados: el daño no se limita a una institución, sino que se propaga al conjunto de la economía nacional.

1.7.2 Petroecuador y Contratos con Odebrecht (2007-2016)

Ecuador fue uno de los países afectado por el escándalo de Odebrecht («Contraloría General del Estado», 2017). La constructora brasileña pagó unos 33,5 millones de dólares en sobornos a funcionarios ecuatorianos a cambio de contratos de Petroecuador (empresa petrolera estatal) valorados en más de 116 millones de dólares. Las vulnerabilidades detectadas incluyeron los sistemas de contratación pública de Petroecuador, que no proporcionaban un registro de auditoría completo para el proceso de evaluación de ofertas. No existían registros digitales de quién accedió a información restringida de la licitación, quién participó en la evaluación técnica, qué criterios de puntuación se aplicaron ni quién aprobó las adendas contractuales que incrementaron los costos. La ausencia de sellado de tiempo seguro y firmas electrónicas en documentos clave permitió la manipulación retroactiva de documentos.

El alcance real del daño solo pudo dimensionarse años después, cuando la Contraloría General del Estado ejerció su potestad auditora. La Contraloría emitió 31 informes finales sobre 15 proyectos en los

que participó Odebrecht, habiendo auditado un universo de USD 4.409,4 millones en contratos y establecido glosas y multas por USD 144,7 millones («Contraloría General del Estado», 2017). El exvicepresidente Jorge Glas fue sentenciado en diciembre de 2017 a seis años de prisión por el delito de asociación ilícita, habiéndose acreditado en juicio. Desde la perspectiva de la auditabilidad de sistemas, este caso es paradigmático: los sistemas de contratación de Petroecuador no generaban trazas inmutables de las decisiones de adjudicación, lo cual habilitaba que un mismo documento pudiera circular con versiones distintas ante distintos actores del proceso, sin que ninguna herramienta tecnológica pudiera detectar la discrepancia en tiempo real.

1.7.3 Sobreprecio en Compras Públicas COVID-19 (2020)

En el contexto de la pandemia de la COVID-19, Ecuador se ha visto afectado por la corrupción relacionada con la compra de suministros médicos de emergencia. La Contraloría General del Estado ha reportado sobreprecios superiores al 9.000 % en la compra de mascarillas, trajes de protección y otros suministros. El uso de procedimientos de emergencia que eludían los controles de contratación pública suprimió las pistas de auditoría. Los sistemas del Portal de Compras Públicas no registraban justificaciones para la contratación directa no competitiva. La ausencia de mecanismos de verificación automatizados que validaran si un proveedor estaba registrado como empresa activa con una trayectoria comprobada permitió la contratación de empresas fantasma. Tampoco se disponía de paneles de monitoreo en tiempo real que alertaran sobre precios atípicos en comparación con los precios de mercado (de Derechos y Justicia, s.f.).

Las cifras de este episodio son concretas y verificables. El Gobierno Nacional destinó USD 664,8 millones para atender la emergencia sanitaria, de acuerdo con cifras oficiales disponibles al corte de mayo de 2021 (y Periodismo de Investigación, 2021). La Contraloría General del Estado, en ejercicio de su mandato constitucional, inició 53 auditorías en apenas tres meses, entre marzo y mayo de 2020, para examinar el uso de esos recursos en contrataciones de insumos

médicos («Contraloría General del Estado», 2020). Los resultados fueron alarmantes: en el caso más documentado, las pantallas faciales de protección adquiridas por el IESS fueron cotizadas a USD 21,53 por unidad frente a un precio de mercado de USD 0,23, lo que representa un sobreprecio del 9.259 %, según el informe con indicios de responsabilidad penal aprobado por el contralor Pablo Celi y remitido a la Fiscalía el 6 de abril de 2020 («Contraloría General del Estado», 2020). Para mayo de 2021, la Fiscalía General del Estado reportó más de 160 investigaciones abiertas vinculadas a estas compras de emergencia (**convoca2021**). Desde el punto de vista de los sistemas de información, este escenario evidencia la consecuencia directa de carecer de validaciones automatizadas en tiempo real: un sistema auditable con reglas de negocio correctamente implementadas habría rechazado automáticamente cualquier cotización que superara un umbral porcentual predefinido respecto al precio de mercado registrado, sin necesidad de intervención humana posterior.

1.8 Lecciones para Sistemas Auditables

Los escándalos mencionados, tanto globales como de Ecuador, son ejemplos de la materialización de sistemas de datos no auditables. A decir:

- ✓ **Registros inmutables:** si los registros pueden modificarse posteriormente, o si los registros no existen o no están completos, puede producirse fraude y no se pueden realizar auditorías. Los registros inmutables deben estar firmados criptográficamente.
- ✓ **cuandol de acceso:** si el acceso a un sistema no se controla ni registra (quién accedió, cuándo y por qué), no es posible saber cuándo un usuario interno utiliza los datos para su propio beneficio.
- ✓ **Flujo de trabajo:** cualquier aprobación o decisión (por ejemplo, un crédito, un contrato, una transferencia) debe realizarse mediante un flujo de trabajo. Cada paso debe registrarse, con los aprobadores y su justificación; se debe generar un registro de evidencia digital.
- ✓ **Análisis en tiempo real:** si los análisis solo se pueden generar

con un desfase (mensual o trimestral), puede producirse un fraude y aumentar hasta que se detecte la anomalía. Debe estar disponible el análisis en tiempo real, así como las alarmas para excepciones.

- ✓ **Segregación de funciones:** si quienes operan los sistemas son los mismos que administran los registros, puede producirse fraude. Los sistemas deben segregar funciones y contar con un sistema de auditoría externa

Estos ejemplos ofrecen un sólido argumento empírico a favor de sistemas sólidos y auditables tanto en el sector público como en el privado, en Ecuador y a nivel global.

1.9 La Respuesta Legislativa: Sarbanes-Oxley Act de 2002

El gobierno estadounidense reaccionó rápidamente a la crisis. En julio de 2002, el Congreso aprobó la Ley de Reforma Contable de las Empresas Públicas y Protección del Inversionista, más conocida como Ley Sarbanes-Oxley (Tunggal & Ruan, 2026). El proyecto de ley fue patrocinado por el senador Paul Sarbanes (demócrata por Maryland) y el representante Michael Oxley (republicano por Ohio).

El proyecto de ley fue aprobado en la Cámara de Representantes por 423 votos a favor y 3 en contra, y en el Senado por 99 votos a favor y 0 en contra.

El presidente estadounidense George Bush promulgó la ley SOX el 30 de julio de 2002, señalando que "La era de los bajos estándares y las ganancias falsas ha terminado; ninguna junta directiva en Estados Unidos está por encima o al margen de la ley". La ley SOX consta de 11 secciones que abordan la supervisión de las empresas públicas, la administración y las firmas de contabilidad («Sarbanes-Oxley Act», n.d.). Las provisiones clave con implicaciones directas para auditabilidad de software incluyen:

- ✓ **Título I - Public Company Accounting Oversight Board (PCAOB):** Estableció el PCAOB como organismo independiente supervisado por la SEC para promover reportes de auditoría precisos, independientes y transparentes. El PCAOB registra

firmas de auditoría, establece estándares de auditoría y ética, inspecciona cumplimiento de SOX y hace cumplir reglas mediante penalidades que incluyen suspensiones, censuras y multas de hasta $2 millones por violación.

- ✓ **Título II - Independencia del Auditor:** Fortalece la independencia de auditores externos limitando conflictos de interés y restringiendo servicios que auditores pueden proporcionar a clientes. Prohíbe que la firma que audita los libros de una compañía pública también haga la contabilidad, audite valuaciones de negocio, diseñe o implemente sistemas de información, proporcione servicios de asesoría de inversiones o banca, o consulte sobre otros temas de gestión.
- ✓ **Sección 302 - Certificación de Controles y Procedimientos:** Requiere que CEO y CFO certifiquen personalmente la precisión de reportes financieros. Específicamente, deben certificar que han revisado el reporte, que no contiene declaraciones materialmente falsas ni omite hechos materiales, que los estados financieros reflejan fielmente la condición financiera, y que son responsables de establecer y mantener controles internos sobre reportes financieros.
- ✓ **Sección 404 - Evaluación de Controles Internos:** Esta es quizás la provisión más controversial e impactante de SOX. Requiere que todos los reportes anuales incluyan un reporte de Control Interno que delinee explícitamente la responsabilidad de la gerencia para mantener una estructura de control interno adecuada, una evaluación de su efectividad, y cualquier deficiencia en esos controles. Los auditores externos independientes deben también atestar la precisión de la declaración de la compañía de que los controles internos están en lugar y son efectivos.

¿Qué implica la Sección 404 para los sistemas de información? Según el informe técnico sobre los requisitos de auditoría de SOX, elaborado por el proveedor de seguridad de datos Imperva, los auditores de SOX revisan áreas como la gestión de usuarios, la autenticación, la segregación de funciones, el control de acceso y, quizás lo más importante, los registros de auditoría en el entorno

de la base de datos. Los sistemas de recopilación y monitorización de registros deben proporcionar un registro de auditoría para todo acceso y uso de datos empresariales confidenciales.

Cumplir con SOX puede ser costoso. Una encuesta realizada en 2008 por la SEC a directivos de empresas públicas reveló que el costo medio anual directo fue de 2,3 millones de dólares.

1.10 Estructura y Provisiones Clave de SOX

Sarbanes-Oxley consta de 11 títulos que establecen nuevos estándares para juntas directivas de compañías públicas estadounidenses, gerencia ejecutiva y firmas de contabilidad pública. La ley es extensa, aproximadamente 66 páginas de texto legal denso y, sus provisiones tocan prácticamente todos los aspectos de gobierno corporativo y reporte financiero. Para propósitos de entender su impacto en auditabilidad de software, varios títulos y secciones específicas son particularmente relevantes:

- ✓ **Título I - Public Company Accounting Oversight Board (PCAOB):** Este título estableció el PCAOB como un organismo nuevo e independiente, supervisado por la SEC, pero operado como entidad sin fines de lucro. El PCAOB tiene autoridad para registrar firmas de contabilidad pública que auditan compañías públicas, establecer estándares de auditoría y ética, inspeccionar firmas de auditoría registradas para evaluar cumplimiento con SOX y otras leyes, y hacer cumplir reglas mediante investigación y acciones disciplinarias incluyendo suspensiones, censuras y multas de hasta $2 millones por violación para individuos y $15 millones para firmas.

1.11 La Respuesta Legislativa Ecuatoriana

Como contexto, se tiene que la Crisis Bancaria fue el catalizador de la reforma, así como Estados Unidos respondió a los escándalos Enron y WorldCom con la Sarbanes-Oxley Act de 2002, Ecuador experimentó su propia crisis sistémica que catalizó reformas legislati-

vas significativas. La crisis bancaria ecuatoriana de 1999-2000, con el colapso del sistema financiero, y el feriado bancario que congeló ahorros de millones de ciudadanos, expuso debilidades fundamentales en gobernanza corporativa y controles internos.

En Ecuador, a diferencia del modelo estadounidense donde la ley SOX abordó múltiples sectores simultáneamente, la respuesta legislativa ecuatoriana fue sectorial y evolutiva. La reforma más análoga a SOX en términos de imponer controles internos rigurosos y responsabilidad de ejecutivos fue la Ley Orgánica de Empresas Públicas (LOEP) promulgada el 16 de octubre de 2009, publicada en Registro Oficial Suplemento No. 48 («Ley Orgánica de Empresas Públicas», 2009).

1.11.1 Ley Orgánica de Empresas Públicas

Artículo 5: Principios de Gestión, el Artículo 5 establece principios que deben regir gestión de empresas públicas:

- ✓ **Rendición de cuentas:** "Las empresas públicas y sus administradores estarán sujetos al escrutinio de los organismos de control correspondientes y deberán implementar sistemas permanentes de rendición de cuentas". Esta provisión es análoga a SOX Sección 302 donde CEO y CFO deben certificar personalmente estados financieros.
- ✓ **Transparencia:** "Las empresas públicas estarán sujetas al principio de transparencia que les obliga a suministrar información veraz, suficiente y oportuna". Similar a SOX Sección 409 que requiere rápida divulgación de cambios materiales.
- ✓ **Eficiencia y calidad:** Requiere "sistemas de gestión integral basados en procesos debidamente documentados", análogo a requisitos de documentación de controles internos de SOX Sección 404.

Artículo 14: Responsabilidad Solidaria del Directorio

"El Directorio responderá solidariamente por las decisiones adoptadas, salvo constancia expresa del voto particular negativo del

respectivo miembro del Directorio".

Esta responsabilidad solidaria es más estricta que SOX, mientras SOX impone responsabilidad específicamente a CEO y CFO, LOEP impone responsabilidad solidaria a todos los miembros del directorio por todas las decisiones corporativas.

Artículo 19: Firma Solidaria de Estados Financieros

"Suscribir los estados financieros en forma solidaria con el responsable financiero, y presentarlos al Directorio".

La firma solidaria entre Gerente General y responsable financiero es directamente análoga a SOX Sección 302, haciendo a ambos ejecutivos legalmente responsables conjuntamente por exactitud de reportes financieros.

Artículo 29: Auditoría Interna Independiente

"Las empresas públicas contarán con una unidad de auditoría interna que dependerá administrativa y funcionalmente del Directorio".

Esta independencia funcional respecto de la administración ejecutiva es análoga a SOX Sección 301 que requiere que un comité de auditoría supervise auditores internos.

Artículo 31: Sistemas de Control Interno

"Las empresas públicas establecerán sistemas permanentes de control interno, compuestos por políticas, normas, métodos y procedimientos estructurados e integrados, a fin de prevenir y mitigar riesgos de gestión".

Esta provisión es directamente análoga a SOX Sección 404 que requiere evaluación y reporte sobre efectividad de controles internos. Requiere que controles sean "estructurados e integrados", implicando sistemas de información que implementen controles programáticamente y generen evidencia auditable.

1.11.2 Contraloría General del Estado: NCI

El Acuerdo 039-CG-2009 emitió las "Normas de Control Interno (NCI) para Entidades del Sector Público", análogas funcionalmente

al Framework COSO utilizado para implementación de SOX 404. Las normas establecen cinco componentes idénticos a COSO: ambiente de control, evaluación de riesgos, actividades de control, información y comunicación, y monitoreo (Contraloría General del Estado, n.d.).

1.11.3 La Norma 500: Información y Comunicación

Esta norma, en la sección 500-01 establece los controles sobre sistemas de información: "Se diseñarán, establecerán y mantendrán controles apropiados de tecnología de información para asegurar la confiabilidad, exactitud y validez de la información procesada".

Esta norma implica requisitos de auditabilidad: sistemas deben capturar datos completos, mantener integridad, y permitir verificación de confiabilidad de información reportada (**normascontrolinterno**).

1.11.4 Código Orgánico Monetario y Financiero (2014)

Para el sector financiero específicamente, el COMYF (Asamblea Nacional del Ecuador, n.d.) consolidó la siguiente regulación:

- ✓ **Artículo 208:** Sistemas de Información: "Las entidades del sistema financiero nacional deberán contar con sistemas de información que permitan mantener registros precisos y completos de las operaciones, garantizando su integridad, confiabilidad y oportunidad". Obligación legal específica de sistemas auditables en sector financiero.
- ✓ **Artículo 445:** Gestión Integral de Riesgos: Requiere sistemas de administración de riesgos que incluyen riesgo operacional, definido como "deficiencias en sistemas de información o controles internos".

Para un mejor entendimiento, en la siguiente tabla 1 consta la comparación del Código SOX Vs. el Marco Ecuatoriano.

Tabla 1:
Comparación SOX vs. Marco Ecuatoriano

Aspecto	**Sarbanes-Oxley (USA)**	**Marco Ecuatoriano**
Trigger	Enron, WorldCom (2001-2002)	Crisis bancaria (1999-2000)
Legislación	SOX Act (2002) - ley única	LOEP (2009), COMYF (2014) - sectorial
Ámbito	Todas las empresas públicas (traded)	LOEP: empresas públicas; COMYF: financiero
Responsabilidad ejecutiva	Sec. 302: CEO/CFO certifican	Art. 19: Gerente/responsable financiero firman solidariamente
Controles internos	Sec. 404: Management evalúa controles	Art. 31: Empresas públicas establecen controles bajo normas CGE
Auditoría interna	Implícito en Sec. 404	Art. 29: Explícitamente requerido
Supervisor	SEC y PCAOB	CGE para público; Superintendencias para privado

Elaboración Propia

1.12 El Caso por el Software Auditable: Evidencia Empírica

Frente a la evidencia de los daños causados por sistemas no auditables, resulta igualmente pertinente documentar lo que la evidencia empírica muestra cuando los controles sí existen. El beneficio del software auditable no es teórico: se mide en pérdidas evitadas, tiempos de detección reducidos y recursos públicos recuperados.

El estudio global más citado en la materia, *Occupational Fraud 2024: A Report to the Nations* de la Association of Certified Fraud Examiners (Association of Certified Fraud Examiners, 2024), analizó 1.921 casos reales de fraude en 138 países e identificó que las organizaciones pierden en promedio el 5 % de sus ingresos anuales por fraude ocupacional, lo que a escala global representa aproximadamente USD 5 billones al año (Association of Certified Fraud Examiners, 2024). El mismo estudio revela que el tiempo mediano que tarda en descubrirse un esquema de fraude es de 12 meses, y que más de la mitad de

los casos están correlacionados con ausencia o debilitamiento de controles internos. Para América Latina y el Caribe, los hallazgos son especialmente preocupantes: la región registra la pérdida mediana más alta por caso a nivel mundial, con USD 250.000 por incidente, superando a todas las demás regiones analizadas.

La misma investigación demuestra, en sentido inverso, el valor de los controles auditables: en organizaciones que contaban con líneas de reporte anónimas y mecanismos formales de denuncia, las pérdidas por fraude fueron un 50 % menores que en aquellas que carecían de ellos. Cuando los controles internos funcionan como primera línea de detección, los fraudes se descubren antes y su impacto económico se reduce de forma significativa. Desde la perspectiva del sector público ecuatoriano, la Contraloría General del Estado ha demostrado este principio de forma directa: sus auditorías sobre contratos Odebrecht en Ecuador, realizadas precisamente porque existían registros auditables que podían ser examinados, permitieron identificar glosas y responsabilidades por USD 144,7 millones en 15 proyectos de infraestructura («Contraloría General del Estado», 2017). Sin sistemas que generen trazas verificables, esa recuperación habría sido imposible. El software auditable, en definitiva, no es un costo de cumplimiento normativo: es el mecanismo que hace posible que el control externo tenga algo sobre qué actuar.

Capítulo 2

FUNDAMENTOS CONCEPTUALES

La auditabilidad del software se refiere a un paradigma fundamental del diseño, desarrollo e implementación de sistemas de información (SI), en el que la disponibilidad de mecanismos que permitan el seguimiento, el examen o la verificación del comportamiento del sistema es un requisito estricto. En un mundo donde las organizaciones deben cumplir con normativas en constante evolución y cada vez más exigentes, como la Ley Sarbanes-Oxley (SOX), el Reglamento General de Protección de Datos (GDPR), la Ley de Portabilidad y Responsabilidad del Seguro Médico (HIPAA) y varias otras, la necesidad de que los sistemas de software sean auditables ha pasado de ser una característica de calidad del software a un requisito obligatorio para el cumplimiento normativo y organizacional.

Según la norma ISO/IEC 25010 (International Organization for Standardization, 2011), la auditabilidad de un sistema de información se define como la capacidad de registrar, preservar y presentar evidencia adecuada y apropiada de sus operaciones para permitir la verificación independiente del cumplimiento del sistema con sus especificaciones, políticas y requisitos regulatorios. Aunque está estrechamente relacionada con otros conceptos como el registro, la monitorización y la observabilidad del sistema, la auditabilidad es una característica distintiva de la calidad del software que abarca las dimensiones arquitectónicas, de diseño, de implementación y operativas, y que debe abordarse a lo largo de todo el ciclo de vida del desarrollo de software.

La Figura 2 ilustra cómo opera la rastreabilidad en la práctica. Considérese una transacción ordinaria en un sistema de información: un usuario autenticado ejecuta una operación, por ejemplo, una transferencia bancaria, una aprobación contractual, una modificación de datos médicos, y ese gesto, aparentemente simple, desencadena una cadena de eventos que atraviesa múltiples capas del sistema. En cada transición, del usuario a la aplicación, de la aplicación a la base de datos, de la base de datos al registro de auditoría, el sistema debe capturar evidencia suficiente para responder con precisión las seis preguntas fundamentales de la rendición de cuentas: quién inició la acción, qué operación se realizó y qué valores cambiaron, cuándo

ocurrió con precisión temporal verificable, desde dónde se originó la solicitud, mediante qué mecanismo técnico se ejecutó, y con qué justificación de negocio fue autorizada.

Figura 2:
Trazabilidad de una transacción de extremo a extremo en un sistema auditable



Nota: Elaboración Propia

Lo que distingue a un sistema auditable de uno que meramente registra actividad es precisamente la completitud y la integridad de esa cadena. Un log que captura el "qué" pero omite el "quién" no permite atribución; uno que registra el "quién" pero no el "cuándo" no permite reconstrucción temporal de eventos; uno que almacena todos los campos, pero permite su modificación posterior no constituye evidencia admisible. La rastreabilidad, en sentido estricto, no es una propiedad de un campo individual del registro: es una propiedad de la cadena completa, desde el actor hasta el almacenamiento inmutable, sin brechas ni puntos de manipulación posible. Cualquier ruptura en esa cadena, sea por diseño deficiente, por omisión en el desarrollo o por vulnerabilidad operativa, constituye exactamente el tipo de vacío que los casos de Filanbanco, Petroecuador y las compras de emergencia COVID-19 expusieron con consecuencias medibles en cientos de millones de dólares.

2.1 Definición Operacional de Auditabilidad y Software Auditable

Se define un sistema de software auditable como un Sistema de Información diseñado, desarrollado, implementado y operado con características inherentes que facilitan, permiten y garantizan la verificación independiente, completa y fiable de su comportamiento, la toma de decisiones, el procesamiento de datos y la conformidad con las especificaciones, políticas y normativas aplicables. Esta definición es intencionadamente extensa y densa; por lo tanto, es necesario aclarar el alcance de cada uno de sus componentes.

Diseñado, desarrollado, implementado y operado implica que la auditabilidad debe considerarse en todas las etapas del ciclo de vida del desarrollo de software. Por ejemplo, en la fase de diseño, la auditabilidad puede implicar el diseño de una arquitectura que facilite la monitorización del sistema, la trazabilidad, la separación de intereses y la modularización. En la fase de desarrollo, la auditabilidad puede referirse a la implementación de prácticas de codificación que promueven la legibilidad, mantenibilidad y testabilidad del código. En la fase de implementación, significa establecer una configuración adecuada del sistema que permita la recopilación de registros y registros de auditoría sin comprometer el rendimiento del sistema. En la fase de operación, puede implicar actividades que garanticen la recopilación, preservación y protección de registros y registros de auditoría.

En el modelo de calidad de software definido por la norma ISO IEC 25010 del año 2011, la auditabilidad aparece como una subcaracterística dentro del atributo de seguridad. Su propósito es apoyar el principio de responsabilidad, es decir, la capacidad de un sistema para dejar claro quién realizó cada acción. Según la norma, la responsabilidad se refiere al grado en que las acciones de una entidad pueden atribuirse de manera única y sin ambigüedades a esa misma entidad. En términos prácticos, esto significa que el software debe permitir rastrear las acciones realizadas, identificar a su autor y contar con evidencias suficientes para evitar dudas, errores o negaciones

sobre lo ocurrido.

En la Tabla 2, se identifican tres pilares fundamentales sobre los que se sustenta el software auditable.

Tabla 2:
Pilares Fundamentales del Software Auditable

Pilar fundamental	**Concepto**
Rastreabilidad (Traceability)	Capacidad de seguir el flujo de ejecución y las transformaciones de datos a través del sistema, estableciendo cadenas de causalidad entre eventos, acciones y resultados (Gotel & Finkelstein, 1994).
No repudio (Non-repudiation)	Garantía criptográfica y procedimental de que las acciones registradas no pueden ser negadas posteriormente por sus autores, proporcionando evidencia irrefutable de la participación en transacciones o eventos (Menezes et al., 1996).
Integridad de registros (Log Integrity)	Protección de los registros de auditoría contra modificación, eliminación o falsificación, asegurando que la evidencia recolectada mantiene su valor probatorio a lo largo del tiempo (Schneier & Kelsey, 1999).

Elaboración Propia

2.2 Definición y Marco Conceptual del Registros de Auditoría

El concepto de "Audit Trail" o Rastro de Auditoría constituye el mecanismo técnico fundamental que posibilita la auditabilidad de un sistema de software. Esta afirmación, aunque simple en superficie, implica una complejidad técnica y conceptual considerable que merece análisis exhaustivo. Un registro de auditoría no es meramente un archivo de log o una tabla de base de datos que registra eventos; es una construcción sociotécnica que media entre requisitos humanos de rendición de cuentas y capacidades técnicas de sistemas computacionales.

El Instituto Nacional de Estándares y Tecnología (NIST), es la agencia del Departamento de Comercio de los Estados Unidos encargada de establecer estándares técnicos, ofrece la que quizás

sea la definición formal más frecuentemente citada de una pista de auditoría, en la Publicación Especial 800-92 del NIST sobre la Guía para la Gestión de Registros de Seguridad Informática. Según el NIST (National Institute of Standards and Technology, 2024), una pista de auditoría es: "Un conjunto de registros que proporcionan evidencia documental de la secuencia de actividades que, en algún momento, han afectado a una operación, procedimiento, evento o dispositivo específico".

Esta definición, aparentemente simple, tiene varios componentes que merecen una explicación más detallada.

En primer lugar, la definición de "conjunto de registros" implica que una pista de auditoría no es un solo registro, sino una colección estructurada de registros. Esta colección debe cumplir con las propiedades de integridad, no puede haber lagunas inexplicables; orden temporal, debe ser posible reconstruir la secuencia cronológica de eventos; y completitud, deben registrarse todos los eventos relevantes.

En segundo lugar, el concepto de "evidencia documental" resalta la naturaleza forense de la pista de auditoría. Los registros no son simplemente informativos; son evidencia que potencialmente podría utilizarse en un entorno legal o regulatorio. Esto tiene implicaciones adicionales: los registros deben ser fiables, deben ser generados por sistemas confiables y verificables, su integridad puede demostrarse; preservados, deben estar protegidos contra alteraciones o destrucción; e interpretables, debe ser posible comprender su significado incluso años después de su creación.

En tercer lugar, la noción de "secuencia de actividades" apunta a la naturaleza temporal y causal del registro de auditoría. Este debe permitir no solo reconstruir qué sucedió, sino también cuándo y en qué orden. En los sistemas distribuidos modernos, donde los eventos pueden ocurrir casi simultáneamente en diferentes nodos, mantener el orden causal supone un desafío técnico significativo. Este desafío ha sido objeto de considerable investigación en sistemas distribuidos, desde el trabajo de Lamport sobre causalidad (Lamport, 1978) hasta los algoritmos contemporáneos para relojes lógicos y vectores de

versión.

2.3 Anatomía de un Registro de Auditoría (Audit Trail)

2.3.1 Sistemas de Logging Estructurado

Los sistemas modernos de logging han evolucionado significativamente desde los simples archivos de texto plano hacia arquitecturas de logging estructurado que permiten el tratamiento de los logs como conjuntos de datos estructurados en lugar de mero texto. Esta evolución responde a la necesidad de analizar grandes volúmenes de eventos en sistemas distribuidos y de alta complejidad.

Un registro de auditoría individual debe capturar información suficiente para responder las preguntas fundamentales de rendición de cuentas: quién, qué, cuándo, dónde, cómo y por qué. La investigación de Elmasri y Navathe (Elmasri & Navathe, 2004) en su obra fundamental sobre sistemas de bases de datos proporciona un framework útil para entender qué constituye un registro de auditoría efectivo en el contexto de sistemas de bases de datos, framework que se generaliza bien a otros tipos de sistemas.

2.3.2 Quién (Who)

Identidad del actor que inició la acción. En sistemas simples, esto puede ser un nombre de usuario. En sistemas complejos, puede requerir información más rica: el identificador de usuario, el rol bajo el cual operaban, la aplicación o servicio que usaron, el proceso o thread específico, y potencialmente información de sesión que permite correlacionar múltiples acciones. Es crítico distinguir entre identidad real (la persona o sistema que autorizó la acción) e identidad efectiva (la cuenta bajo la cual se ejecutó la acción), particularmente en contextos donde ocurre delegación o suplantación de identidad.

2.3.3 Qué (What)

La acción específica realizada. Esto debe ser lo suficientemente específico para ser significativo. No es suficiente registrar 'acceso a datos'; el Rastro de Auditoría debe registrar qué datos específicos fueron accedidos, qué operación se realizó (lectura, escritura, actualización,

eliminación), qué valores cambiaron (antes y después para updates), y cuál fue el resultado de la operación (éxito, fallo, parcialmente exitosa). En sistemas transaccionales, esto puede incluir el ID de transacción que permite agrupar múltiples operaciones relacionadas.

2.3.4 Cuando (When)

Timestamp preciso del evento. En sistemas distribuidos, esto presenta desafíos sutiles. Los relojes de diferentes nodos pueden estar desincronizados; un timestamp de un solo nodo puede ser insuficiente para determinar ordenamiento causal. Soluciones modernas utilizan Network Time Protocol (NTP) para sincronización de relojes, pero incluso con NTP, precisión es típicamente solo en el rango de milisegundos. Para ordenamiento más preciso, se utilizan técnicas como Lamport timestamps o vector clocks que capturan causalidad lógica en lugar de tiempo físico absoluto.

2.3.5 Dónde (Where)

Ubicación desde la cual se originó la acción. En sistemas tradicionales, esto puede ser dirección IP y hostname. En ambientes cloud modernos, puede incluir zona de disponibilidad, región geográfica, identificador de instancia de contenedor, y espacio de nombres de Kubernetes. Para acciones iniciadas por usuarios, puede incluir información de geolocalización. Para acciones de sistemas automatizados, debe incluir identificador del job o servicio que la inició.

2.3.6 Cómo (How)

El mecanismo o método mediante el cual se realizó la acción. ¿Fue a través de una interfaz de usuario web? ¿Una API REST? ¿Una llamada directa a base de datos? ¿Un job batch?. Esta información es relevante para investigaciones de seguridad porque patrones de acceso anómalos frecuentemente se revelan en el "cómo", un usuario que normalmente accede datos a través de la aplicación web repentinamente haciendo consultas SQL directas puede indicar compromiso de credenciales.

2.3.7 Por qué (Why)

Esta es la dimensión más difícil de capturar automáticamente. Idealmente, un registro de auditoría capturaría la justificación de negocio para una acción. En algunos sistemas regulados, particularmente en salud, los usuarios pueden ser requeridos de proporcionar una razón explícita al acceder ciertos datos sensibles. Sin embargo, en la mayoría de los sistemas, el "por qué" debe ser inferido del contexto, el propósito de negocio típico de la aplicación o servicio que generó la acción.

En el ecosistema .NET, el framework de Log denominado Serilog (Serilog Contributors, n.d.) representa el estado del arte en logging estructurado. A diferencia de las bibliotecas tradicionales de logging que se basan en cadenas de formato interpoladas, Serilog utiliza message templates, un DSL (Domain Specific Language) que extiende las cadenas de formato de .NET con parámetros nombrados. Por ejemplo, en lugar de escribir:

```
1    log.Information("Usuario " + userId + " proces
        ó orden " + orderId);
```

Serilog permite escribir:

```
1    log.Information("Usuario {UserId} procesó
        orden {OrderId}", userId, orderId);
```

Esta aparente diferencia sintáctica tiene implicaciones profundas. En el segundo caso, Serilog serializa userId y orderId como propiedades estructuradas del evento de log, permitiendo consultas posteriores como "mostrar todos los eventos donde OrderId = 12345" sin necesidad de hacer "parsing" de texto o expresiones regulares.

La arquitectura de Serilog se basa en el concepto de "sinks" (depósitos o sumideros) que representan destinos para los eventos de log: archivos, consola, bases de datos, servicios en la nube, sistemas SIEM (Security Information and Event Management), entre otros. Esta arquitectura permite que la misma aplicación envíe logs a múltiples destinos simultáneamente sin modificar el código de negocio

2.4 Requisitos de Integridad y No Repudio

Sin embargo, un registro de auditoría solo es útil si es confiable, en el sentido de que puede usarse como evidencia de lo que realmente sucedió. Para ser confiable, un registro de auditoría debe cumplir con las propiedades de seguridad de integridad y no repudio.

Integridad: La integridad de un registro de auditoría se refiere a la propiedad de que, una vez registrados los datos, estos no pueden alterarse, eliminarse ni falsificarse. Esta propiedad es importante porque un atacante o persona con acceso interno que pueda modificar un registro de auditoría puede usar esta capacidad para ocultar sus huellas, enmascarando así comportamientos maliciosos.

Implementar la integridad del Registros de Auditoría no es sencillo. El enfoque más sencillo consiste en registrar el registro de auditoría en medios de escritura única y lectura múltiple (WORM) (Veritas-Chain, n.d.). Si bien conceptualmente simple, este enfoque suele ser poco práctico por razones de coste y operativas. Un enfoque más sofisticado consiste en utilizar técnicas criptográficas para dificultar la manipulación de los datos del Registros de Auditoría. Un esquema clásico, consiste en que cada entrada del registro de auditoría sea una función de la entrada anterior; en concreto, cada entrada contiene un hash criptográfico de la entrada anterior. Esto crea una cadena de entradas enlazadas: si un atacante intenta modificar una entrada, la cadena se rompe, y este hecho es detectable (DataSunrise, 2025).

Figura 3:
Mecanismo de firma digital para garantía de no repudio en registros de auditoría



Nota: Elaboración Propia

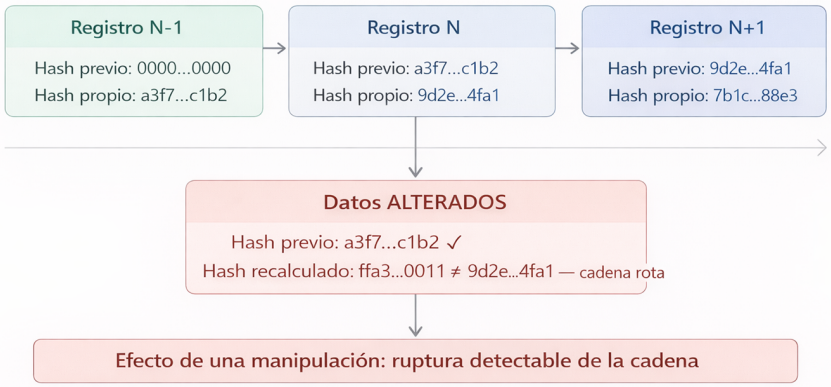
La Figura 3 descompone el mecanismo mediante el cual un sistema auditable convierte una acción humana en evidencia irrefutable. El proceso no comienza en el momento del registro, sino antes: en la verificación de identidad del actor. Sin autenticación robusta en el punto de entrada, cualquier registro posterior carece de valor probatorio, porque no es posible vincular la acción a una persona o entidad responsable de forma inequívoca.

Una vez que el actor autenticado ejecuta la operación, el sistema genera una huella digital del evento mediante una función hash criptográfica, típicamente SHA-256 en implementaciones modernas, que produce un valor único e irreproducible a partir del contenido exacto del registro. La propiedad matemática fundamental de estas funciones es que cualquier alteración del contenido, por mínima que sea, produce un hash completamente distinto. Esto significa que la huella no puede falsificarse sin que la falsificación sea detectable. A continuación, un sello de tiempo certificado por una Autoridad de Sellado de Tiempo, bajo el estándar RFC 3161, ancla el evento a un momento cronológico verificable por terceros, impidiendo que el registro sea antedatado o postdatado. El resultado final es un registro firmado que garantiza simultáneamente tres propiedades:

autenticidad, en tanto solo el titular de la clave privada pudo generar esa firma; integridad, en tanto cualquier modificación del contenido invalida la firma; y no repudio, en tanto el actor no puede negar posteriormente haber ejecutado la acción sin contradecir la evidencia criptográfica.

Este mecanismo es exactamente el que estaba ausente en los sistemas de contratación de Petroecuador durante el caso Odebrecht: sin firmas digitales en los documentos de adjudicación ni sellos de tiempo certificados, un mismo contrato podía circular con versiones distintas ante distintos actores del proceso, y ninguna herramienta técnica permitía determinar cuál versión era la original ni quién había autorizado cada modificación.

Figura 4:
Cadena de hashes criptográficos para integridad de registros



Nota: Elaboración Propia

La Figura 4 materializa el concepto abstracto de integridad mediante su implementación más directa: la cadena de hashes criptográficos. El principio de funcionamiento es elegante en su simplicidad y robusto en sus consecuencias. Cada registro de auditoría no es una entrada independiente, sino un eslabón que incorpora en su propio contenido la huella digital del registro anterior. De este modo, el sistema construye una dependencia matemática acumulativa: el

registro N solo puede existir con ese valor de hash si el registro N-1 tenía exactamente ese contenido, y el registro N-1 solo puede tener ese hash si el N-2 no fue alterado, y así sucesivamente hasta el primer registro de la cadena.

La parte inferior de la figura ilustra la consecuencia inmediata de cualquier intento de manipulación. Si un actor con acceso privilegiado modifica los datos del registro N, suprimiendo una transacción comprometedora, alterando un monto, cambiando una fecha, el hash que ese registro recalcula ya no coincide con el hash que el registro N+1 almacena como referencia de su predecesor. La cadena se rompe en ese punto de forma detectable y objetiva, sin necesidad de testimonio humano ni de comparación contra una copia externa. El sistema se audita a sí mismo.

Esta propiedad tiene implicaciones directas para el diseño de software auditable en el sector público ecuatoriano. En el caso de Filanbanco, la existencia de dos contabilidades paralelas fue posible precisamente porque los registros podían modificarse sin dejar rastro verificable de la modificación. Un esquema de cadena de hashes habría convertido cada alteración en una anomalía matemáticamente demostrable, detectable por cualquier auditor con acceso al log, sin depender de testigos internos ni de investigaciones posteriores de años de duración. La diferencia entre un sistema que registra y un sistema que garantiza integridad no es de grado: es la diferencia entre evidencia y mera información.

2.5 Registros de Auditoría en Bases de Datos

Las bases de datos merecen atención especial en cualquier discusión de registros de auditoría porque contienen frecuentemente los activos más valiosos de una organización y son objetivo principal tanto de atacantes externos como de amenazas internas.

Los sistemas de gestión de bases de datos (DBMS) mantienen inherentemente ciertos tipos de logs para propósitos de recuperación ante fallos. El transaction log o write-ahead log (WAL) (Mohan et al., 1992) es un mecanismo estándar en virtualmente todos los

DBMS que soportan transacciones ACID. Este log registra todas las modificaciones hechas a la base de datos de manera que el DBMS puede recuperarse de crashes mediante replay o rollback de transacciones. Sin embargo, los transaction logs (Grokipedia, 2026) están diseñados para recuperación, no para auditoría, y típicamente no capturan toda la información requerida para propósitos de Rastro de Auditoría.

Para auditoría efectiva, los DBMS deben extender un logging transaccional básico con información adicional. Se proponen tres enfoques principales para implementar registros de auditoría en bases de datos, cada uno con diferentes trade-offs entre completitud, performance y complejidad de implementación:

2.5.1 Auditoría basada en triggers

Este enfoque implementa database triggers que automáticamente se disparan cuando ocurren operaciones de INSERT, UPDATE o DELETE en tablas designadas como críticas (IBM, n.d.). Los triggers registran información relevante de auditoría, típicamente el usuario que hizo el cambio, timestamp, los valores antiguos y nuevos de datos modificados, en tablas de auditoría separadas. Este enfoque tiene la ventaja de ser transparente para aplicaciones; no se requiere modificación de código de aplicación. Sin embargo, tiene varias desventajas. Primero, los triggers añaden overhead a cada operación en tablas auditadas, potencialmente impactando performance significativamente. Segundo, los triggers pueden ser deshabilitados por usuarios con privilegios suficientes, creando una potencial vulnerabilidad de seguridad. Tercero, los triggers capturan solo cambios que ocurren a través de operaciones DML normales; no capturan necesariamente cambios hechos mediante operaciones DDL, operaciones bulk, o acceso directo a archivos de base de datos.

2.5.2 Auditoría mediante transaction logs

Este enfoque extiende el transaction log existente del DBMS (Grokipedia, 2026) para incluir información adicional de auditoría como el identificador de usuario, el terminal o aplicación desde el cual se ejecutó la transacción, y potencialmente otra metadata

contextual. Este enfoque tiene overhead de performance relativamente bajo porque el DBMS ya está manteniendo un transaction log. Sin embargo, presenta desafíos. Los transaction logs son típicamente escritos en formato binario propietario optimizado para recuperación rápida, no para lectura humana. Extraer información de auditoría requiere herramientas especializadas que entiendan el formato de log específico del DBMS. Además, transaction logs típicamente se archivan o eliminan después de cierto período para gestión de espacio, potencialmente violando requisitos de retención de registros de auditoría (IBM, n.d.).

2.5.3 Auditoría a nivel de aplicación

Este enfoque implementa lógica de auditoría explícitamente en el código de aplicación. Cada vez que la aplicación realiza una operación crítica, también registra explícitamente un evento de auditoría («Audit trail», n.d.). Este enfoque proporciona máxima flexibilidad, la aplicación puede registrar exactamente lo que es relevante desde perspectiva de negocio, puede capturar contexto de negocio rico que la base de datos no conoce, y puede implementar lógica de auditoría sofisticada que va más allá de simplemente registrar cambios de datos. Sin embargo, requiere disciplina y vigilancia rigurosa en desarrollo. Es fácil olvidar agregar logging de auditoría a nuevas características o pasar por alto los casos edge. Si la aplicación es comprometida, un atacante puede potencialmente evitar o manipular auditoría a nivel de aplicación.

En la práctica, organizaciones sofisticadas frecuentemente utilizan combinaciones de estos enfoques, implementando defense-in-depth para registros de auditoría. Por ejemplo, triggers de base de datos podrían capturar todos los cambios como fallback, mientras que código de aplicación registra información contextual más rica para flujos normales.

Capítulo 3

MARCOS REGULATORIOS Y NORMATIVOS

3.1 El Estándar IEEE 1028-2008

El estándar IEEE 1028-2008, titulado formalmente "IEEE Standard for Software Reviews and Audits" (Institute of Electrical and Electronics Engineers, 2008), constituye el marco normativo técnico más ampliamente reconocido para la conducción de auditorías de software. Publicado por el IEEE Computer Society y aprobado por la IEEE Standards Board el 16 de junio de 2008, este estándar representa décadas de conocimiento consolidado sobre cómo evaluar sistemáticamente productos y procesos de software.

IEEE 1028-2008 es una revisión de IEEE Std 1028-1997 (Institute of Electrical and Electronics Engineers, 1997), que a su vez revisó IEEE Std 1028-1988. Esta genealogía de revisiones refleja la evolución continua de mejores prácticas en ingeniería de software y la necesidad de actualizar estándares para reflejar cambios en tecnologías, metodologías y contexto regulatorio. La versión 2008 hizo cambios mayores en estructura y contenido comparada con la versión 1997, reflejando experiencia acumulada y retroalimentación de la industria sobre la aplicación del estándar.

El propósito declarado de IEEE 1028 es proporcionar requisitos mínimos aceptables para revisiones sistemáticas de software, donde "sistemático" incluye participación de equipo, resultados documentados, y procedimientos definidos. IEEE 1028 no especifica cómo determinar si una revisión o auditoría es necesaria, esa decisión queda relegada a la organización. Tampoco especifica qué hacer con los resultados de una revisión o auditoría, el estándar se enfoca puramente en el proceso de conducir la revisión o auditoría misma.

3.2 ISO/IEC 25010: Modelo de Calidad de Software

El estándar ISO/IEC 25010 (International Organization for Standardization, 2011), parte de la serie SQuaRE (Systems and software Quality Requirements and Evaluation), proporciona un modelo comprensivo de calidad de software que, aunque no enfocado exclusivamente en auditabilidad, incluye características de calidad que son fundamentales para software auditable. Publicado originalmente

en 2011 como reemplazo de ISO/IEC 9126 y actualizado en 2023, ISO/IEC 25010 refleja décadas de investigación académica e industrial sobre cómo medir y evaluar calidad de software.

ISO/IEC 25010 organiza calidad de software en dos modelos complementarios: un modelo de calidad del producto y un modelo de calidad en uso. Esta dualidad reconoce que la calidad debe evaluarse tanto desde una perspectiva técnica (propiedades del producto de software mismo) como desde una perspectiva de usuario (efectividad del producto cuando se usa en contexto real). Para propósitos de entender la auditabilidad, el modelo de calidad del producto es primariamente relevante.

3.3 GDPR: El Reglamento General de Protección de Datos

El Reglamento General de Protección de Datos de la Unión Europea (GDPR por sus siglas en inglés) (European Union, 2016a), que entró en vigencia el 25 de mayo de 2018, representa quizás la regulación de privacidad más completa e influyente jamás promulgada. Con 99 artículos y 173 considerandos que abarcan 88 páginas de texto legal denso, GDPR establece un régimen regulatorio que ha servido como modelo para la legislación de privacidad en múltiples jurisdicciones globales y, ha forzado cambios fundamentales en cómo las organizaciones alrededor del mundo procesan datos personales.

GDPR se aplica a cualquier organización que procesa datos personales de individuos ubicados en la Unión Europea, independientemente de dónde esté ubicada la organización misma. Esta aplicación extraterritorial ha dado a GDPR influencia global. Ejemplo, una compañía estadounidense que opera un sitio web accesible desde Europa, o una empresa china que vende productos a consumidores europeos, o un proveedor de servicios cloud que almacena datos de clientes europeos, todos están potencialmente sujetos a GDPR. Esta jurisdicción extraterritorial es explícita en el Artículo 3, que especifica que GDPR se aplica al procesamiento de datos personales en contexto de actividades de un establecimiento de un controlador o procesador en la UE, así como al procesamiento de datos personales de individuos

que están en la UE por un controlador o procesador no establecido en la UE, donde las actividades de procesamiento están relacionadas con ofrecer bienes o servicios a tales individuos o, monitorear su comportamiento.

3.4 HIPAA y los Controles de Auditoría

La HIPAA Security Rule, específicamente en el estándar 45 CFR 164.312(b) (U.S. Department of Health and Human Services, n.d.-a), establece requisitos explícitos para controles de auditoría. Las entidades cubiertas y asociados de negocio deben "implementar mecanismos de hardware, software y/o procedimientos que registren y examinen la actividad en los sistemas de información que contienen o usan información de salud electrónica protegida (ePHI)".

El Artículo 33 del GDPR impone un plazo de 72 horas para notificar brechas de datos a las autoridades supervisoras, mientras que HIPAA requiere notificación dentro de 60 días. Estas diferencias en los plazos de notificación tienen implicaciones significativas para el diseño de sistemas de monitoreo y detección de incidentes, requiriendo capacidades de análisis en tiempo real o casi tiempo real.

3.5 Principios Fundamentales y el Concepto de Accountability

El artículo 5 del GDPR (European Union, 2016b) enumera los 7 principios clave para el tratamiento de datos personales. Estos principios no constituyen directrices, sino obligaciones legales, y su incumplimiento puede conllevar multas cuantiosas. Los 7 principios son: licitud, lealtad y transparencia; limitación de la finalidad; minimización de datos; exactitud; limitación del almacenamiento; integridad y confidencialidad; y rendición de cuentas.

El principio de rendición de cuentas (artículo 5(2)) es muy interesante desde la perspectiva de la auditabilidad del software: "El responsable del tratamiento se encargará del cumplimiento del apartado 1 y podrá demostrarlo. El elemento clave aquí es la "capacidad de demostrar". No basta con que su organización

cumpla con el GDPR; también debe poder demostrarlo mediante documentación, procesos y sistemas auditables.

El principio de rendición de cuentas se implementa en el GDPR mediante el artículo 30, que exige que los responsables y encargados del tratamiento mantengan registros de las actividades de tratamiento, que deben contener el nombre y los datos de contacto del responsable del tratamiento, los fines del tratamiento, Una descripción de las categorías de interesados y de datos personales; las categorías de destinatarios a quienes se han divulgado o se divulgarán los datos; cualquier transferencia de datos personales a terceros países; los plazos previstos para la supresión de las diferentes categorías de datos; y una descripción general de las medidas de seguridad técnicas y organizativas.

Esto significa que los responsables del tratamiento que utilizan sistemas de software complejos necesitan un registro de auditoría completo. No basta con tener registros estáticos de los datos que se procesan; los responsables del tratamiento deben poder demostrar, basándose en registros auditables, qué procesamiento se ha realizado, quién lo ha autorizado, bajo qué base legal y qué garantías se aplican.

3.6 Protección de Información de Salud en Estados Unidos

En Estados Unidos, la Ley de Portabilidad y Responsabilidad del Seguro Médico (HIPAA) de 1996 (U.S. Department of Health and Human Services, n.d.-c), establece estándares nacionales para la seguridad de la información sanitaria electrónica. Si bien se adoptó más de 20 años antes del GDPR, la HIPAA fue igualmente pionera en su momento al exigir garantías técnicas para proteger los datos sensibles. En el contexto de la auditabilidad, la Norma de Seguridad de la HIPAA reviste especial interés.

Publicada en 2003, la Norma de Seguridad es una regulación federal que describe una serie de estándares y especificaciones de implementación que deben implementarse para proteger la confidencialidad, integridad y disponibilidad de la información médica electrónica protegida (ePHI), según lo exige la Ley de Portabilidad y

Responsabilidad del Seguro Médico de 1996. Las entidades cubiertas y sus socios comerciales deben adherirse a los estándares que detallan cómo proteger la información médica electrónica protegida (ePHI) según la Norma de Seguridad.

45CFR164.312(b) (U.S. Department of Health and Human Services, n.d.-b) es la especificación para los Controles de Auditoría. El texto de la especificación es conciso, pero su significado es sustancial: "Implementar hardware, software y/o mecanismos de procedimiento que registren y examinen la actividad en los sistemas de información que contienen o utilizan información médica electrónica protegida". Esta es una especificación de implementación abordable, pero en la práctica, casi todas las organizaciones la implementan. Sin registros de auditoría, es casi imposible demostrar que se está protegiendo adecuadamente la ePHI.

La guía del Department of Health and Human Services sobre la Security Rule menciona sobre qué deben capturar los controles de auditoría. Los registros de auditoría deben registrar acceso a ePHI, intentos de acceso negados, cambios o eliminaciones de ePHI, y acceso a los registros de auditoría mismos. La guía enfatiza que los registros de auditoría son tanto para detectar brechas de seguridad como para investigarlas después de que ocurren (U.S. Department of Health and Human Services, n.d.-c).

El Reglamento General de Protección de Datos (GDPR) de la Unión Europea introduce requisitos específicos de auditabilidad en sus artículos 30 (registros de actividades de procesamiento), 33 (notificación de brechas de seguridad) y 35 (evaluaciones de impacto en protección de datos), estableciendo que las organizaciones deben poder demostrar el cumplimiento mediante evidencia documental y técnica (European Union, 2016b).

En el contexto latinoamericano, regulaciones como la Ley de Protección de Datos Personales de Argentina (Ley 25.326), la Ley Federal de Protección de Datos Personales en Posesión de los Particulares de México (2010) y normativas específicas de instituciones financieras como la Superintendencia de Bancos en Ecuador establecen requisitos

similares de trazabilidad y auditabilidad para sistemas que procesan datos sensibles (Saltor, 2013).

3.7 Protección de Datos Personales (GDPR) en Ecuador

La protección de datos personales en Ecuador busca que la información de las personas sea tratada con respeto y responsabilidad. Aunque el GDPR es una norma europea, muchos de sus principios influyen en la legislación ecuatoriana y en la forma en que las organizaciones manejan datos personales. Esto implica recopilar solo la información necesaria, usarla de manera transparente, protegerla con medidas de seguridad adecuadas y permitir que las personas conozcan, corrijan o soliciten la eliminación de sus datos. En la práctica, se trata de generar confianza, evitar abusos y asegurar que el uso de la información esté alineado tanto con la ley local como con buenas prácticas reconocidas a nivel internacional.

3.7.1 Influencia del GDPR europeo en Ecuador

El Reglamento (UE) n.º 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, entró en vigor el 25 de mayo de 2018 (European Union, 2016b) y es el reglamento de protección de datos más amplio e importante a nivel mundial. A pesar de ser una ley europea, el GDPR tiene implicaciones para las entidades ecuatorianas que tratan datos de residentes de la UE o que tienen una oficina o establecimiento en la UE.

El ámbito de aplicación territorial del GDPR se define en el artículo 3. El artículo 3.1 establece que el GDPR se aplica al tratamiento de datos personales "en el contexto de las actividades de un establecimiento de un responsable o de un encargado del tratamiento en la Unión, independientemente de que el tratamiento tenga lugar o no en la Unión". El Artículo 3.2 establece que el GDPR también se aplica al tratamiento de datos personales de interesados en la Unión por parte de un responsable o encargado del tratamiento no establecido en la Unión, cuando las actividades de tratamiento estén relacionadas con la "oferta de bienes o servicios, independientemente de si se requiere el pago del interesado, a dichos interesados en la

Unión" o con la "supervisión de su comportamiento, siempre que este tenga lugar dentro de la Unión".

Este efecto extraterritorial implica que una empresa ecuatoriana que ofrezca servicios de comercio electrónico dirigidos al mercado europeo, que gestione datos personales de residentes de la UE o que supervise la actividad en internet de usuarios residentes en Europa, estará obligada a cumplir con el GDPR y, por lo tanto, a cumplir con todas sus medidas de protección de datos, que incluyen medidas concretas de auditoría y rendición de cuentas.

3.7.2 Requisitos de Auditoría y Accountability del GDPR

Uno de los conceptos relevantes en los que se basa el GDPR es el de la rendición de cuentas (responsabilidad proactiva). Según el Artículo 5(2): "El responsable del tratamiento dará cumplimiento de las disposiciones establecidas en el apartado 1 y podrá demostrarlo ("rendición de cuentas proactiva")". En esencia, esto significa que las organizaciones no solo deberán adherirse a los principios de protección de datos, sino que también deberán demostrarlo con pruebas que lo respalden.

Los registros de las actividades de tratamiento son ahora un requisito formal según el artículo 30(1) del GDPR: cada responsable del tratamiento y, en su caso, su representante, mantendrán un registro de las actividades de tratamiento bajo su responsabilidad. El responsable del tratamiento o su representante pondrá este registro a disposición de la autoridad de control previa solicitud. Dichos registros contendrán toda la información siguiente: el nombre y los datos de contacto del responsable del tratamiento y, en su caso, del corresponsable del tratamiento, su representante y el delegado de protección de datos; los fines del tratamiento; una descripción de las categorías de interesados y de las categorías de datos personales; las categorías de destinatarios a los que se han comunicado o se comunicarán los datos personales, incluidos los destinatarios en terceros países u organizaciones internacionales; en su caso, las transferencias de datos personales a un tercer país o una organización internacional, incluida la identificación de dicho tercer

país u organización internacional y, en el caso de las transferencias a que se refiere el artículo 49(1), párrafo segundo, la documentación de las garantías adecuadas; cuando sea posible, los plazos previstos para la supresión de las diferentes categorías de datos; cuando sea posible, una descripción general de los aspectos técnicos y Medidas de seguridad organizativas a que se refiere el Artículo 32(1).

Estos documentos constituyen registros administrativos que registran cómo una organización gestiona la información personal. Deben ser escritos, posiblemente electrónicos, y ponerse a disposición de las autoridades que lo soliciten (Artículo 30.4). Este requisito no se aplica a las organizaciones con menos de 250 empleados, a menos que el tratamiento implique un riesgo para los derechos y libertades de los interesados, no sea ocasional o incluya categorías especiales de datos (Artículo 30.5).

El Artículo 32 aborda la seguridad del tratamiento y establece que el responsable y el encargado del tratamiento deben implementar medidas técnicas y organizativas apropiadas para garantizar un nivel de seguridad adecuado al riesgo. El Artículo 32(1)(d) establece que esto debe incluir "procedimientos para la comprobación, evaluación y valoración periódicas de la eficacia de las medidas técnicas y organizativas para garantizar la seguridad del tratamiento". Esto implica auditorías de seguridad periódicas, pruebas de control y documentación de los resultados.

El artículo 33 exige que las notificaciones de violaciones de datos personales se envíen a la autoridad de control en un plazo de 72 horas desde su conocimiento, siempre que sea probable que suponga un riesgo para los derechos y libertades de las personas físicas. El artículo 34 exige que las violaciones se comuniquen a los interesados si es probable que supongan un alto riesgo para sus derechos y libertades. Estos requisitos de notificación implican que las organizaciones deben poder detectar rápidamente las violaciones mediante sistemas de seguimiento, determinar su alcance mediante el análisis de registros de auditoría y documentar exhaustivamente el incidente (hechos, consecuencias y medidas de mitigación) (artículo 33 (5)).

3.7.3 Marco Constitucional Ecuatoriano de Protección de Datos

Ecuador reconoce la protección de datos personales como derecho constitucional. La Constitución de la República del Ecuador de 2008 (Registro Oficial, 2008), establece en su Artículo 66, numeral 19, el derecho a la protección de datos de carácter personal, incluyendo acceso y decisión sobre información y datos de este carácter, así como su correspondiente protección. El mismo artículo especifica que la recolección, archivo, procesamiento, distribución o difusión de datos o información personal requieren autorización del titular o mandato de ley.

El Artículo 92 de la Constitución establece la acción de hábeas data, mecanismo constitucional mediante el cual toda persona puede conocer de la existencia y acceder a documentos, datos genéticos, bancos o archivos de datos personales e informes que sobre sí misma consten en entidades públicas o privadas. También puede solicitar actualización, rectificación, eliminación o anulación de datos que fueren erróneos o afecten ilegítimamente sus derechos.

3.7.4 Ley Orgánica de Protección de Datos Personales (2021)

Ámbito de Aplicación: El artículo 3 de la LOPDP (Registro Oficial, 2021), establece que la ley se aplicará al tratamiento automatizado o parcialmente automatizado de datos personales, así como al tratamiento no automatizado de datos destinados a formar parte de un archivo. La ley se aplicará a los responsables y encargados del tratamiento de datos en Ecuador, dondequiera que se realice el tratamiento (artículo 3, apartado a), así como a los responsables y encargados no establecidos en Ecuador, cuando la actividad de tratamiento tenga por objeto ofrecer bienes o servicios a los interesados en Ecuador o esté relacionada con la vigilancia de su comportamiento en territorio ecuatoriano (artículo 3, apartado b). La disposición reproduce esencialmente el texto sobre la aplicación extraterritorial del GDPR y lo adapta a la jurisdicción ecuatoriana.

Principios de Protección de Datos: El artículo 8 de la

LOPDP (Registro Oficial, 2021), detalla los principios que deben observarse en el tratamiento de datos personales, como: (a) licitud (tratamiento lícito y respeto de los derechos del interesado); (b) lealtad y transparencia (transparencia de la información para el interesado); (c) limitación de la finalidad (finalidad específica, explícita y legítima); (d) minimización de datos (adecuada, pertinente y limitada a lo necesario); (e) exactitud (exacta y actualizada); (f) limitación del almacenamiento (durante el tiempo necesario); y (g) seguridad (medidas técnicas y organizativas adecuadas).

El requisito clave para la auditoría es el artículo 8, apartado f), que exige a los responsables del tratamiento: "aplicar principios y directrices para garantizar y poder demostrar que el tratamiento de datos personales cumple con esta Ley y que podrán demostrar dicho cumplimiento ante la Autoridad de Protección de Datos". Al igual que el artículo 5.2 del GDPR, este artículo exige claramente la presentación de pruebas de dicho cumplimiento.

Registro de Actividades de Tratamiento: Los responsables y encargados del tratamiento deberán mantener registros de las actividades de tratamiento bajo su responsabilidad.

El artículo 35 de la LOPDP (Registro Oficial, 2021), establece que cada responsable y encargado del tratamiento debe mantener un registro de las operaciones de tratamiento bajo su responsabilidad. La información que debe contener el registro es similar a la establecida en el artículo 30 del GDPR: identidad del responsable, fines del tratamiento, descripción de los interesados y de los datos, destinatarios, transferencias a terceros países, plazos de conservación y descripción de las medidas de seguridad (artículo 35, apartados 1-8).

El artículo 35, apartado 2, establece que el registro debe presentarse a la Autoridad de Protección de Datos cuando se lo solicite. Esta obligación recae sobre todas las organizaciones, excepto aquellas con menos de veinte empleados (menor que el GDPR), a menos que el tratamiento presente un riesgo para los derechos de los interesados, no sea ocasional o incluya datos sensibles (artículo 35, apartado 3).

Seguridad del Tratamiento: Según el artículo 34 de la LOPDP

(Registro Oficial, 2021), los responsables y encargados del tratamiento deben adoptar medidas técnicas, organizativas y legales adecuadas para proteger los datos personales. En particular, dichas medidas deben evitar la pérdida, alteración, tratamiento o acceso no autorizado; garantizar la confidencialidad, integridad, disponibilidad y resiliencia de los sistemas de tratamiento; restablecer la disponibilidad y el acceso a los datos de forma oportuna en caso de incidente; y verificar, evaluar y valorar periódicamente la eficacia de las medidas técnicas y organizativas (artículo 34, apartados a-d).

El apartado d establece los mismos requisitos que el artículo 32(1)(d) del GDPR y, por lo tanto, exige que las organizaciones ecuatorianas tengan la obligación legal de mantener sistemas auditables, y que dichas auditorías se puedan realizar periódicamente para validar los controles establecidos.

Notificación de Violaciones de Seguridad: Los artículos 37 y 38 de la LOPDP (Registro Oficial, 2021), describen las disposiciones sobre notificación de violaciones, similares a las del GDPR. El artículo 37 estipula que los responsables del tratamiento deben notificar a la Autoridad de Protección de Datos la violación de datos en un plazo de 72 horas desde su conocimiento, si esta pudiera suponer un riesgo para los derechos del interesado. El artículo 38 estipula que se debe informar al interesado afectado de la violación cuando sea probable que represente un alto riesgo para sus derechos.

Para cumplir con estas obligaciones de notificación, las organizaciones deberán ser capaces de: identificar las violaciones (mediante sistemas de seguimiento y análisis de registros de auditoría), evaluar el alcance y los daños causados por las violaciones (realizando análisis forenses de registros) y registrar: los hechos que rodean la violación; las categorías y el número de interesados (y, cuando sea posible, los registros) afectados; los efectos de la violación; y las medidas ya adoptadas o propuestas (artículo 37, apartados 1-4).

Autoridad de Protección de Datos Personales: El artículo 55 de la LOPDP (Registro Oficial, 2021), establece la Autoridad de Protección de Datos, adscrita a la Defensoría del Pueblo, con

autonomía administrativa, operativa y financiera. La Autoridad de Protección de Datos tendrá competencias de supervisión, investigación y sanción en materia de protección de datos personales, similares a las de las autoridades europeas de protección de datos previstas en el GDPR.

El artículo 57 define las funciones de la Autoridad, que incluyen: supervisar y garantizar la aplicación del Reglamento, atender reclamaciones, realizar investigaciones, imponer multas o sanciones coercitivas, emitir advertencias, órdenes y amonestaciones, aprobar certificaciones de protección de datos, aprobar normas corporativas vinculantes, aprobar o aprobar los criterios de certificación, establecer cláusulas tipo de protección de datos, prestar asesoramiento, mantener una lista de las operaciones de tratamiento sujetas al requisito de una evaluación de impacto de la protección de datos y fomentar la elaboración de códigos de conducta (artículo 57, letras a a l).

El artículo 62 define el régimen de sanciones administrativas para las infracciones, que se califican como leves, graves y muy graves, según los criterios establecidos en el artículo 60. Estas sanciones incluyen una advertencia; Multa del 0,1 % al 4 % de la facturación anual total del ejercicio anterior, hasta un máximo de 250 000 USD; suspensión temporal de las actividades de tratamiento de datos; y prohibición permanente del tratamiento de datos (artículo 62). Este régimen de sanciones es proporcionado, aunque menos extenso que el del GDPR (hasta 20 millones de euros o el 4 % de la facturación anual total, la cantidad que sea mayor).

3.7.5 Reglamento de la LOPDP (2023)

El Reglamento a la Ley Orgánica de Protección de Datos Personales fue emitido mediante Decreto Ejecutivo No. 678 de 30 de marzo de 2023 (Registro Oficial, 2023), publicado en Registro Oficial Suplemento 267 de 7 de abril de 2023. El Reglamento desarrolla aspectos operativos de la LOPDP y proporciona guía más detallada sobre implementación de obligaciones.

El Artículo 18 del Reglamento especifica contenido mínimo del registro de actividades de tratamiento con mayor detalle que la

LOPDP. El Artículo 19 establece que el registro debe actualizarse al menos anualmente o cuando ocurran cambios significativos en actividades de tratamiento. El Artículo 20 especifica formato del registro, permitiendo formato electrónico o físico, pero requiriendo que sea accesible, organizado, y comprensible.

Los artículos 23-27 del Reglamento son más específicos y se refieren a las medidas técnicas, entre ellas: la seudonimización y el cifrado de datos personales, la capacidad de garantizar la confidencialidad, integridad, disponibilidad y resiliencia permanentes de los sistemas y servicios de tratamiento, la capacidad de restablecer la disponibilidad y el acceso a los datos personales de forma oportuna en caso de incidente físico o técnico, y un procedimiento para probar, evaluar y evaluar periódicamente la eficacia de las medidas técnicas y organizativas para garantizar la seguridad del tratamiento (Registro Oficial, 2023) (artículo 23).

El artículo 24 se centra en las auditorías de seguridad. Indica que los responsables y encargados del tratamiento deben "evaluar periódicamente la eficacia de las medidas técnicas y organizativas implantadas y conservar la documentación pertinente, que estará a disposición de la Autoridad". Propone que se realice al menos una auditoría de seguridad al año, pero esta frecuencia podrá aumentar en función del riesgo del tratamiento.

3.8 Implicaciones para Sistemas Auditables en Ecuador

En este contexto, el marco legal ecuatoriano en materia de protección de datos personales, particularmente la Ley Orgánica de Protección de Datos Personales y su Reglamento, define un conjunto de obligaciones específicas relacionadas con la auditabilidad para los responsables del tratamiento de datos personales. Estas obligaciones buscan asegurar que las actividades de tratamiento puedan ser verificadas, trazadas y justificadas ante autoridades competentes, garantizando así la transparencia, la rendición de cuentas y el cumplimiento efectivo de los principios legales establecidos:

✓ **Obligación Legal de Mantener Registros de Tratamiento:**

Si bien el registro de las actividades de tratamiento es una acción recomendada, el art. 35 de la LOPDP lo exige por ley. El responsable del tratamiento debe establecer un mecanismo para registrar las actividades de tratamiento de datos personales. Se debe registrar toda la información sobre los datos personales tratados, la finalidad, el responsable del tratamiento y la fecha.

- ✓ **Obligación de Auditar Seguridad Regularmente:** El artículo 34 (d) de la LOPDP y el artículo 24 del Reglamento establecen explícitamente que las medidas de seguridad se auditarán periódicamente. En otras palabras, es un requisito legal contar con medidas para verificar y probar periódicamente estos controles y registrar los resultados. Los sistemas de información deben diseñarse teniendo esto en cuenta, en lugar de dificultarlo o imposibilitarlo.
- ✓ **Obligación de Detectar y Documentar Violaciones:** Los artículos 37 y 38 de la LOPDP establecen un plazo de 72 horas para notificar una violación. Esto no se puede lograr a menos que una organización cuente con un sistema capaz de detectar una brecha de seguridad a tiempo, mediante la monitorización y revisión continuas de los datos de las pistas de auditoría. Las organizaciones deberían invertir en sistemas de Gestión de Eventos e Información de Seguridad (SIEM) o similares que analicen los registros en tiempo real y alerten cuando se produzca una actividad sospechosa.
- ✓ **Obligación de Rendición de Cuentas:** El artículo 8 (f) de la LOPDP prioriza la rendición de cuentas de forma proactiva, y se espera que las organizaciones demuestren su cumplimiento ante la Autoridad, si así se les solicita. Esto incluiría el mantenimiento de pistas de auditoría completas; registros de las consideraciones de diseño para la protección de datos; registros de los consentimientos; evidencia del cumplimiento de los diversos derechos de los interesados (acceso, rectificación, supresión); y registros de las evaluaciones de impacto sobre la privacidad realizadas.
- ✓ **Separación de Datos de Auditoría**: Si bien la LOPDP no lo exige directamente, el principio de rendición de cuentas sugiere que

la integridad de las pistas de auditoría también es esencial. Puede ser una buena práctica (y es posible que la Autoridad adopte más adelante esta interpretación) mantener los registros de auditoría en un sistema independiente que imponga un acceso limitado y garantice su integridad y encriptación.

Para un entendimiento más sucinto en la Tabla 3 consta la Comparación de GDPR Vs LOPDP.

Tabla 3:
Comparación GDPR vs. LOPDP

Aspecto	**GDPR (UE)**	**LOPDP (Ecuador)**
Entrada en vigor	25 mayo 2018	26 mayo 2021
Base legal	Reglamento (UE) 2016/679	Ley Orgánica (RO 459, 26-may-2021)
Accountability	Art. 5(2) - responsabilidad proactiva	Art. 8(f) - responsabilidad proactiva (idéntico)
Registro actividades	Art. 30 - obligatorio	Art. 35 - obligatorio
Umbral excepción registro	<250 empleados (con excepciones)	<20 trabajadores (con excepciones)
Auditorías seguridad	Art. 32(1)(d) - verificación regular	Art. 34(d) - verificación regular (idéntico)
Notificación violaciones	Art. 33 - 72 horas	Art. 37 - 72 horas (idéntico)
Comunicación a titulares	Art. 34 - cuando alto riesgo	Art. 38 - cuando alto riesgo (idéntico)
Sanciones máximas	€20M o 4 % facturación global	$250,000 o 4 % facturación local
Autoridad supervisión	Autoridades nacionales independientes	Autoridad de Protección de Datos (adscrita Defensoría del Pueblo)
Extraterritorialidad	Art. 3 - oferta de bienes/servicios o monitoreo en UE	Art. 3 - oferta de bienes/servicios o monitoreo en Ecuador

Elaboración Propia

Capítulo 4

ARQUITECTURA DE SOFTWARE AUDITABLE

4.1 Modelo de Cinco Capas para Sistemas Auditables

La construcción de sistemas de software verdaderamente auditables requiere más que simplemente agregar logging a una aplicación existente. Requiere una aproximación arquitectónica sistemática que considere la auditoría como una preocupación de primera clase que permea todas las capas del sistema. El modelo de cinco capas propuesto en la literatura contemporánea de ingeniería de software proporciona un framework conceptual robusto para entender y diseñar sistemas auditables completos.

Este modelo estratifica las responsabilidades de auditoría en cinco capas funcionales distintas pero interconectadas, mismas que se muestran en la Tabla 4:

Tabla 4:
Capas de los Sistemas Auditables

Nro.	Capa
1	Capa de Captura: Interceptación de Eventos Relevantes
2	Capa de Enriquecimiento: Añadiendo Contexto Semántico
3	Capa de Almacenamiento: Persistencia Inmutable y Append-Only
4	Capa de Protección: Garantías Criptográficas
5	Capa de Análisis: Consulta y Forense

Elaboración Propia

Cada capa tiene responsabilidades específicas, interfaces bien definidas con capas adyacentes, y requisitos no-funcionales particulares. La separación clara de responsabilidades entre capas facilita no solo el diseño y la implementación sino también el razonamiento sobre propiedades de seguridad, la evolución del sistema, y la verificación de cumplimiento regulatorio.

4.2 Taxonomía de Eventos Auditables

Una de las decisiones arquitectónicas más críticas en el diseño de sistemas auditables consiste en determinar qué eventos específicos requieren ser registrados en el Rastro de Auditoría. Sin un marco

metodológico sistemático para tomar esta decisión, los equipos de desarrollo frecuentemente adoptan aproximaciones ad-hoc que resultan en cobertura de auditoría inconsistente y desbalanceada. Este enfoque improvisado típicamente produce dos problemas opuestos: ciertos aspectos del sistema generan volúmenes excesivos de logs con bajo valor informativo (sobre-auditoría), mientras componentes críticos para seguridad o cumplimiento carecen de logging adecuado (sub-auditoría). La consecuencia es ruido informacional que obscurece eventos verdaderamente importantes y gaps de cobertura que impiden investigaciones forenses efectivas o demostración de cumplimiento regulatorio.

La literatura académica en ingeniería de software y las guías de cumplimiento regulatorio han propuesto múltiples taxonomías para clasificar eventos auditables de manera sistemática. El NIST Special Publication 800-92 "Guide to Computer Security Log Management" (Kent & Souppaya, 2006) proporciona una de las taxonomías más completas, categorizando eventos según su fuente generadora (sistema operativo, aplicación, dispositivos de seguridad, dispositivos de red) y su naturaleza funcional. El framework COBIT 2019 de ISACA (ISACA, n.d.) clasifica eventos auditables según procesos de gobierno de TI que requieren evidencia de ejecución. ISO/IEC 27001:2013 (International Organization for Standardization, 2013) en su Anexo A.12.4 especifica categorías de eventos que deben ser sujeto de logged para controles de seguridad de información.

4.3 Clasificación Primaria: Eventos Técnicos versus Eventos de Negocio

Una categorización fundamental ampliamente adoptada distingue entre dos tipos principales de eventos según su nivel de abstracción y audiencia objetivo: eventos técnicos y eventos de negocio.

4.3.1 Eventos Técnicos

Capturan operaciones de infraestructura y sistema a bajo nivel de abstracción. Esta categoría incluye:

- ✓ **Operaciones de base de datos:** Se trata de operaciones DML que se realizaron en los objetos de la base de datos (IBM, n.d.) (por ejemplo, las operaciones INSERT, UPDATE, DELETE o SELECT que accedieron a una tabla con datos sensibles o críticos). Para cualquier operación DML, un evento técnico de base de datos puede proporcionar la siguiente información: el texto de la sentencia SQL (y sus parámetros para evitar que los datos sensibles se incluyan en los registros, si es necesario), el nombre de las tablas y columnas a las que se accedió, el número de filas afectadas, el ID de la transacción de la que formaba parte la operación (para que se puedan ver todas las operaciones dentro de la misma unidad atómica de trabajo), el nivel de aislamiento de la transacción y el tiempo que tardó en ejecutarse la sentencia. Estos sistemas de auditoría de bases de datos suelen funcionar con disparadores, funciones nativas de auditoría de bases de datos como la Auditoría Unificada de Oracle y la Auditoría de SQL Server, o un sistema de Captura de Datos de Cambio que lee el archivo de registro de transacciones de forma asíncrona.
- ✓ **Invocaciones de APIs y servicios web:** Esto incluye llamadas a las API REST (la ruta URL y el método HTTP), llamadas a procedimientos remotos mediante SOAP o gRPC (el nombre del procedimiento), llamadas a microservicios (incluidas las llamadas de comunicación entre procesos), etc. Estos eventos incluyen información como: la API o el servicio específico llamado; los parámetros pasados (como encabezados HTTP, parámetros de consulta, el cuerpo de la solicitud serializado o un hash del cuerpo si es demasiado grande para transmitirse en el evento); el código de estado de la respuesta (como el código de estado HTTP o el código de estado de gRPC); el tiempo de procesamiento de la llamada en milisegundos; cualquier error o excepción generada por la llamada, etc. En las arquitecturas de microservicios, estos eventos suelen incluir identificadores de correlación o seguimiento (como los identificadores de correlación y seguimiento especificados en el estándar de contexto de seguimiento del W3C) para que pueda rastrear las llamadas en múltiples servicios.

- ✓ **Cambios de configuración de sistema:** Se incluyen: Cambios en los archivos de configuración (XML, JSON, YML), cambios en las variables de entorno, cambios en la configuración de la aplicación (datos de configuración) en una base de datos o servicio, cambios en la configuración de seguridad. Los registros de cambios de configuración incluyen: valores antiguos y nuevos de los elementos de configuración (diferencia de configuración), quién aprobó el cambio, quién lo aplicó, a qué hora se aplicó, si se reinició un servicio.
- ✓ **Actividad de red y comunicaciones:** Como las conexiones de red activas (conexiones TCP con IP y puerto de origen y destino), las solicitudes HTTP/HTTPS, incluyendo los detalles del protocolo de enlace TLS (versión del protocolo, conjunto de cifrado negociado, si el certificado fue validado), la transferencia de grandes cantidades de datos (número de bytes enviados y recibidos) y eventos relacionados con el firewall (si una conexión fue permitida o denegada según una regla determinada). Al ejecutar contenedores en un clúster con una malla de servicios, este tipo de eventos suele ser generado por el proxy sidecar, que actúa como proxy de interceptación para todo el tráfico que entra y sale.
- ✓ **Operaciones de autenticación y autorización:** Como: los intentos de inicio de sesión exitosos o fallidos, incluyendo la dirección IP desde la que se intentó la conexión y el agente de usuario (con la geolocalización que se puede inferir de la dirección IP), la elevación de privilegios (como ejecutar sudo en Linux o solicitar la elevación del UAC en Windows), los cambios en la pertenencia a grupos o roles, la emisión y validación de tokens de autenticación (como JWT, aserciones SAML o tokens de acceso OAuth) y el resultado de la evaluación de las políticas de autorización (si el acceso se permitió o denegó según RBAC, ABAC u otros motores como Open Policy Agent). La mayoría de las soluciones modernas de gestión de identidades y accesos (I&A), como Azure AD, Okta o Auth0, pueden generar este tipo de eventos automáticamente y proporcionar API o interfaces de streaming para exportarlos a un SIEM.

✓ **Eventos de sistema operativo:** Tales como: la creación y finalización de procesos, incluyendo la línea de comandos completa y las variables de entorno del proceso, cambios en los permisos o la propiedad de archivos, la instalación o eliminación de software o paquetes, modificaciones del kernel o eventos desencadenados por tareas programadas o trabajos tipo "cron". Existen numerosas herramientas especializadas para la auditoría a nivel de sistema operativo, como Linux auditd, el registro de eventos de Windows o agentes EDR (Detección y Respuesta de Endpoints) como CrowdStrike o SentinelOne, que pueden generar este tipo de eventos con un alto nivel de detalle.

Este tipo de eventos de seguridad son técnicos y solo tienen sentido para ingenieros de infraestructura y sistemas, arquitectos de seguridad y nube, o analistas de un Centro de Operaciones de Seguridad (SOC). Sin embargo, no existe una correlación directa entre estos eventos de bajo nivel y actividades que puedan comprenderse a nivel empresarial, como acciones que los usuarios o analistas de negocio puedan comprender.

4.3.2 Eventos de Negocio

Capturan operaciones significativas desde la perspectiva del dominio de negocio de la aplicación, modelando acciones que tienen consecuencias en procesos de negocio, estado de entidades de dominio, o cumplimiento de reglas de negocio. Esta categoría incluye: Transacciones financieras, operaciones de gestión de órdenes, workflows de procesos de negocio, acceso a información sensible o confidencial, operaciones de gestión de usuarios y control de acceso, eventos de auditoría regulatoria.

4.4 Principios Fundamentales de Diseño

La construcción de software auditable requiere adherencia a principios arquitectónicos específicos que, aunque complementan principios generales de diseño de software, tienen énfasis y matices particulares en el contexto de auditoría. Estos principios deben ser considerados desde las primeras etapas de diseño arquitectónico y

mantenerse vigilantemente durante todo el ciclo de vida del sistema.

4.5 Segregación de Responsabilidades de Auditoría

En auditoría, la segregación de responsabilidades parte de una idea simple: ninguna persona ni componente del sistema debería poder hacerlo todo. No es solo una buena práctica de diseño de software, sino una exigencia para que los registros de auditoría sean realmente confiables. Por eso, las tareas de generar eventos auditables, almacenar los registros, analizarlos y administrar el sistema de auditoría deben estar repartidas entre componentes distintos, con responsabilidades claras y accesos bien controlados. De esta forma se evita que un mismo actor pueda alterar evidencias, ocultar actividad o auditarse a sí mismo.

En sistemas monolíticos tradicionales, es común que el mismo código de aplicación que implementa lógica de negocio también sea responsable de generar logs, determinar qué loggear, y potencialmente incluso gestionar dónde se almacenan los logs. Esta amalgamación de responsabilidades crea riesgos de seguridad: código comprometido puede manipular tanto la lógica de negocio como los registros de auditoría que se supone deben rastrear esa lógica.

En arquitectura verdaderamente segregada, el código de aplicación emite eventos de auditoría a través de una interfaz bien definida (por ejemplo, un "message bus", una API de logging centralizada, o un "event stream") pero no tiene conocimiento ni control sobre cómo esos eventos son posteriormente procesados, almacenados o analizados. Un subsistema de auditoría separado, potencialmente ejecutándose en infraestructura completamente separada, administrado por equipos diferentes, con bases de datos separadas, es responsable de recibir estos eventos, validarlos, enriquecerlos, almacenarlos inmutablemente, y proporcionar capacidades de análisis.

Esta segregación debe extenderse a la infraestructura. Bases de datos que almacenan registros de auditoría deben estar en servidores separados de bases de datos transaccionales. Storage de archivos para logs debe estar en sistemas de archivos separados con controles de

acceso independientes. Idealmente, la infraestructura de auditoría debe estar en una zona de red separada con reglas de firewall estrictas que permiten flujo unidireccional de eventos de auditoría desde el sistema principal hacia el sistema de auditoría, pero no viceversa.

4.6 Inmutabilidad de Registros de Auditoría

La inmutabilidad es un paradigma de diseño en software que establece que los datos escritos nunca se modifican. Por lo tanto, se ha convertido en una opción de diseño popular para facilitar el razonamiento, lograr concurrencia sin bloqueos y proporcionar reproducibilidad. En el caso de la auditoría, la inmutabilidad no es una opción de diseño, sino un requisito que debe cumplirse para que los registros de auditoría sean confiables.

El siguiente paso es utilizar un almacenamiento de datos más duradero e intrínsecamente inmutable. Por ejemplo, los dispositivos de almacenamiento WORM (Write-Once Read-Many), que se utilizan habitualmente con fines de cumplimiento normativo (como en los sectores financiero y sanitario), almacenan datos en soportes que no se pueden modificar una vez escritos. Algunos sistemas de almacenamiento empresarial pueden activarse en "modo de cumplimiento normativo" para garantizar la inmutabilidad, incluso contra los administradores de almacenamiento. El almacenamiento en la nube de Amazon S3 ofrece "Bloqueo de objetos", que permite que los objetos sean inmutables durante un tiempo determinado.

4.7 Paradigma Append-Only para Registros de Auditoría

La inmutabilidad es equivalente al principio de solo anexar. Si la inmutabilidad indica que no se puede cambiar nada, el modelo de solo anexión indica que lo único que se puede hacer es anexar nuevos datos al final de la lista, lo que prohíbe las inserciones en cualquier otro punto, aunque no alteren el contenido anterior.

El modelo de solo anexión presenta algunas características interesantes para razonar sobre la procedencia temporal de un registro de auditoría. Suponiendo un registro puramente de solo anexión, el orden

físico de los registros debería corresponder al orden lógico de escritura de registros. Esto facilita la verificación de que las marcas de tiempo dentro del registro sean monótonas (o al menos no decrecientes, para tener en cuenta eventos simultáneos).

4.8 Principio de Mínimo Privilegio para Acceso a Auditoría

El principio de mínimo privilegio, que usuarios y procesos deben tener solo los permisos mínimos necesarios para realizar sus funciones legítimas, es fundamental en seguridad de sistemas. En el contexto de auditoría, este principio tiene aplicaciones específicas: minimizar quién puede generar registros de auditoría, quién puede leerlos, quién puede configurar el sistema de auditoría, y quién puede administrar retención y archivado.

4.9 Captura y enriquecimiento de eventos

La determinación de qué eventos deben ser auditados es quizás la decisión arquitectónica más crítica en el diseño de un sistema auditable. Auditar muy poco resulta en registros de cobertura que pueden ser explotados o que impiden investigaciones forenses efectivas. Auditar demasiado genera volúmenes masivos de datos con bajo valor informativo, obscurece eventos verdaderamente importantes, consume recursos de almacenamiento y procesamiento innecesariamente, y puede incluso crear problemas de cumplimiento con regulaciones de privacidad que requieren minimización de datos.

Los marcos de cumplimiento proporcionan guías sobre categorías de eventos que típicamente requieren auditoría. El NIST Special Publication 800-53 en su familia de controles AU (Audit and Accountability) (National Institute of Standards and Technology, 2024) especifica múltiples categorías de eventos auditables. El GDPR en su Artículo 30 requiere mantener registros de actividades de procesamiento, efectivamente requiriendo auditoría de todas las operaciones que involucran datos personales.

Sin embargo, estas guías son inherentemente genéricas. Su aplicación a un sistema específico requiere análisis cuidadoso del dominio de

negocio, amenazas de seguridad anticipadas, requisitos regulatorios específicos del sector, y objetivos de auditoría de la organización. Este análisis debe ser sistemático, documentado, y mantenido a lo largo del ciclo de vida del sistema.

4.10 Almacenamiento Seguro de Registros de Auditoría

El almacenamiento de registros de auditoría requiere consideraciones especiales que van más allá de las bases de datos transaccionales convencionales. (Schneier & Kelsey, 1999) propusieron el concepto de "untrusted computer base audit logging", donde incluso si el sistema auditado es comprometido, los registros de auditoría permanecen íntegros y verificables.

4.10.1 Captura de Contexto Enriquecido

Los eventos de auditoría en su forma más básica capturan qué ocurrió (una operación específica se ejecutó) y cuándo (timestamp). Sin embargo, para que los registros de auditoría sean verdaderamente útiles para análisis forense, investigación de incidentes, o demostración de cumplimiento, deben capturar contexto mucho más rico sobre las circunstancias en las cuales ocurrió el evento, no solo la operación realizada, sino el contexto completo en el que ocurrió. Esto incluye información sobre el usuario, sesión, dispositivo, ubicación geográfica y contexto de negocio (McCarroll & Rowley, 1979).

4.10.2 Contexto de Identidad y Autenticación

El componente más fundamental del contexto de auditoría es identidad: quién (o qué) inició la acción siendo auditada. En el nivel más básico, esto es el identificador de usuario del actor autenticado. Sin embargo, en sistemas sofisticados, identidad tiene múltiples facetas que todas pueden ser relevantes para auditoría.

Primero, hay la distinción entre identidad real e identidad efectiva. La identidad real es la persona o sistema que inició la sesión original. La identidad efectiva es la identidad bajo la cual se ejecutó una operación específica, que puede diferir si ocurrió suplantación (impersonation) o delegación. Por ejemplo, un administrador de sistema

podría autenticarse con sus propias credenciales, pero luego ejecutar comandos "como" otro usuario para troubleshooting. Los registros de auditoría deben capturar tanto la identidad real del administrador como la identidad efectiva del usuario siendo impersonado.

Segundo, en sistemas que implementan control de acceso basado en roles (RBAC) o control de acceso basado en atributos (ABAC), el rol o conjunto de permisos bajo el cual operaba el usuario cuando realizó la acción es información de auditoría crítica. Un usuario puede tener múltiples roles; registros de auditoría deben registrar qué rol estaba activo para una operación particular. Esto es particularmente importante para detectar escalación de privilegios, situaciones donde un usuario abuse de un rol privilegiado para realizar acciones no relacionadas con las responsabilidades de ese rol.

4.11 Append-Only Databases

4.11.1 Concepto y Principios Fundamentales

Las bases de datos append-only (solo-añadir) representan un paradigma de almacenamiento de datos que impone una restricción arquitectónica fundamental: una vez que los datos han sido escritos al almacenamiento, no pueden ser modificados ni eliminados. Las únicas operaciones permitidas son "append" (añadir nuevos registros al final de la secuencia de datos) y "read" (leer registros existentes). Esta restricción, aunque aparentemente limitante, proporciona propiedades valiosas para sistemas que requieren inmutabilidad, auditabilidad, y rastreo histórico completo de cambios.

El concepto de append-only no es nuevo en ciencias de la computación. Los "transaction logs" de bases de datos relacionales tradicionales (write-ahead logs en PostgreSQL, redo logs en Oracle, transaction logs en SQL Server) han operado siempre en modo append-only por razones de rendimiento y recuperación (performance & recovery), escribir secuencialmente al final de un archivo es significativamente más rápido que actualizar registros en ubicaciones específicas del disco, y mantener secuencia inmutable de operaciones permite recuperarse ante fallos del sistema mediante el "replay" del

log. Lo que diferencia las bases de datos append-only modernas es que elevan este concepto de mecanismo interno de implementación a modelo de datos visible y explotable por aplicaciones.

4.11.2 Inmutabilidad y Temporal Semantics

La inmutabilidad es una propiedad matemática donde un valor, una vez creado, nunca cambia. En contexto de estructuras de datos inmutables, las operaciones que conceptualmente "modifican" una estructura en realidad "crean" una nueva versión, dejando la versión anterior intacta. Los lenguajes de Programación Funcional como Haskell y Clojure construyen sus modelos de programación enteramente sobre inmutabilidad, proporcionando beneficios tales como: razonamiento simplificado sobre comportamiento de programa, sin efectos colaterales, concurrencia sin necesidad de bloqueos, no hay competencia de acceso (race conditions) si los datos nunca cambian y, capacidad de mantener múltiples versiones de datos simultáneamente para diferentes contextos de ejecución.

Las bases de datos Append-only aplican estos principios de inmutabilidad al nivel de persistencia de datos. En lugar del modelo CRUD tradicional (Create, Read, Update, Delete), implementan el modelo CRQ (Create, Read, Query historical states). Cuando una aplicación necesita "actualizar" un registro, la base de datos no modifica el registro existente, sino que añade un nuevo registro que representa el nuevo estado, frecuentemente con metadata que lo vincula al estado anterior. El estado actual de una entidad es simplemente la versión más reciente en la secuencia de versiones.

Esta aproximación naturalmente soporta "temporal semantics", que es la capacidad de consultar no solo el estado actual de datos sino también estados históricos en puntos temporales específicos. Así, se distingue dos tipos principales de dimensiones temporales en estas bases de datos:

- ✓ **Transaction time (tiempo de transacción):** Registra cuándo un dato fue almacenado en la base de datos. Es controlado completamente por el sistema de base de datos y es inmutable, una vez que un registro es comprometido (committed) con un

sello de tiempo de transacción (transaction timestamp), ese sello nunca cambia. Esto permite responder preguntas como: "¿Qué información tenía la base de datos en su estado del 15 de enero de 2026 a las 14:30?"

- ✓ **Valid time (tiempo válido o tiempo de vigencia):** Registra el período durante el cual un hecho es verdadero en el mundo real que se está modelando. Es controlado por la aplicación (la aplicación especifica desde y hasta cuándo un dato es válido) y puede diferir del transaction time. El "Valid time" permite responder preguntas como: "¿Qué empleados trabajaban en el departamento de Marketing durante el segundo trimestre de 2025?" independientemente de cuándo esa información fue registrada en la base de datos.

Las bases de datos bitemporales soportan ambas dimensiones simultáneamente, permitiendo consultas (queries) complejas que combinan criterios de transaction time y valid time. Las bases de datos Append-only naturalmente soportan transaction time porque cada registro tiene timestamp inmutable de cuándo fue escrito. Soportar valid time requiere que la aplicación incluya campos de fecha de inicio/fin de validez en registros.

4.11.3 Log-Structured Storage Engines

Los módulos de almacenamiento (storage engines) basados en estructuras de log (Log-Structured Merge-Trees o LSM-trees) implementan naturalmente semántica append-only. Los LSM-trees fueron propuestos originalmente por (O'Neil et al., 1996) en su artículo "The Log-Structured Merge-Tree (LSM-Tree)" y han sido adoptados ampliamente en bases de datos modernas diseñadas para alto tráfico de escritura.

La arquitectura de LSM-tree opera en múltiples niveles. Primero, las escrituras entrantes se acumulan en memoria en una estructura ordenada llamada "memtable". Luego, cuando la memtable alcanza un tamaño límite, es enviada al disco como archivo inmutable llamado "SSTable" (Sorted String Table). Esta operación de escritura en disco se conoce como "flush" y es una operación append-only, el archivo

SSTable se escribe secuencialmente y nunca se modifica después. Las lecturas deben consultar la memtable en memoria más todos los SSTables en disco, fusionando resultados para encontrar la versión más reciente de cada key.

Periódicamente, el sistema ejecuta un proceso de compactación donde múltiples SSTables se fusionan en SSTables nuevos de nivel superior, eliminando versiones obsoletas de keys y reclamando espacio. Aunque la compactación elimina versiones antiguas, esto ocurre mediante creación de nuevos archivos inmutables, los SSTables originales permanecen inmutables hasta que son reemplazados atómicamente por SSTables compactados.

A continuación, ejemplos de bases de datos que utilizan LSM-trees:

- ✓ **Apache Cassandra:** Base de datos distribuida NoSQL diseñada para escalabilidad horizontal masiva. Cassandra utiliza LSM-trees (implementados en su CommitLog y SSTables) y naturalmente mantiene múltiples versiones de datos mediante timestamps. Cada mutación en Cassandra incluye timestamp hasta con microsegundos, y cuando existen múltiples versiones de una misma columna, Cassandra retorna la versión con timestamp más reciente. Para auditoría, Cassandra puede configurarse con "Time To Live" (TTL) muy largo o infinito, manteniendo versiones históricas indefinidamente. Las consultas pueden especificar un timestamp para recuperar estado histórico de datos en ese punto temporal.
- ✓ **Apache HBase:** Base de datos NoSQL orientada a columnas construida sobre Hadoop Distributed File System (HDFS). HBase mantiene múltiples versiones de cada celda (intersección de row key, column family, column qualifier) con timestamps. El número de versiones retenidas es configurable por column-family. HBase permite consultas con timestamp específico para recuperar estado histórico.
- ✓ **RocksDB:** Biblioteca de "key-value store" desarrollada por Facebook (ahora Meta) como un fork optimizado de LevelDB de Google. RocksDB implementa LSM-tree con optimizaciones agresivas de rendimiento. Aunque en sí mismo no expone direc-

tamente múltiples versiones a aplicaciones (por defecto retorna solo versión más reciente de cada key), es usado como motor de almacenamiento por múltiples sistemas de más alto nivel que sí soportan versionamiento, incluyendo CockroachDB y TiDB.

4.11.4 Event Stores Especializados

Los "Event Stores" son bases de datos diseñadas específicamente para Event Sourcing pattern, donde la primitividad fundamental es el evento inmutable. Estos sistemas son inherentemente append-only porque los eventos, una vez escritos, nunca deben modificarse.

EventStoreDB es un ejemplo de base de datos de código abierto diseñada específicamente para Event Sourcing Pattern, fue desarrollada por Event Store Ltd. y organiza eventos en "streams", donde cada stream es una secuencia ordenada de eventos relacionados. Las operaciones soportadas son:

- ✓ **Append:** Consiste en añadir uno o más eventos al final de un stream. Para manejar concurrencia, esta operación puede incluir una versión esperada del stream. El append solo se completa con éxito si la versión actual coincide con esa versión esperada, lo que permite implementar control de concurrencia optimista y evita que escrituras concurrentes se traslapen entre sí.
- ✓ **Read:** Leer eventos de un stream, opcionalmente desde una posición específica (a la que se podría llamar "versión") y con dirección hacia adelante (forward) o hacia atrás (backward). Por ejemplo, ReadStreamAsync("order-123", StreamPosition.Start, 100) indica que debe leer los primeros 100 eventos del stream "order-123".
- ✓ **Subscribe:** Suscribirse a un stream para recibir notificaciones cuando nuevos eventos son añadidos, permitiendo un procesamiento reactivo. EventStoreDB soporta "catch-up subscriptions" que primero reproducen eventos históricos y luego continúan recibiendo nuevos eventos en tiempo real.

EventStoreDB garantiza que los eventos sean completamente

inmutables, no existe API para modificar o eliminar eventos. El único mecanismo de "eliminación" es el truncamiento de stream, que elimina eventos anteriores a una posición específica, pero esto requiere permisos especiales y genera evento de metadata que documenta dicho truncamiento (proporcionando Rastro de Auditoría de la eliminación misma).

Apache Kafka (Apache Software Foundation, n.d.) aunque típicamente categorizado como plataforma de streaming distribuido, Kafka puede servir efectivamente como base de datos append-only para casos de uso de auditoría. Kafka organiza mensajes en "topics" particionados. Cada partición es una secuencia ordenada append-only de registros con offsets monótonamente crecientes. Los mensajes se escriben al final de las particiones y son inmutables después de su escritura.

Kafka soporta compactación de log, mecanismo donde retiene solo el valor más reciente para cada key dentro de un topic, eliminando versiones antiguas. Sin embargo, para auditoría, la compactación típicamente se deshabilita, manteniendo todos los mensajes indefinidamente. Kafka puede retener terabytes o petabytes de datos históricos, limitado solo por capacidad de almacenamiento del cluster.

Kafka proporciona APIs que permiten leer mensajes desde offset arbitrario, facilitando replay histórico. Por ejemplo, una aplicación puede resetear su posición a offset 0 para reproducir todos los mensajes desde el inicio del topic, útil para reconstruir estado o realizar análisis histórico.

4.11.5 Implementación en Bases de Datos Relacionales

Aunque SQL tradicional permite UPDATE y DELETE sin restricción, es posible implementar semántica append-only sobre bases de datos relacionales mediante patrones de diseño y controles de seguridad apropiados (IBM, n.d.).

Temporal Tables en SQL Server. SQL Server 2016 introdujo soporte nativo para tablas temporales system-versioned conforme al estándar SQL:2011 (Microsoft, 2025b). Una tabla temporal consiste de dos tablas físicas:

✓ **Current table (tabla actual):** que contiene filas actuales con dos columnas adicionales de tipo datetime2 que definen el período de validez: “SysStartTime” (timestamp cuando la fila se volvió válida) y “SysEndTime” (timestamp cuando la fila dejó de ser válida, o valor máximo de datetime2 para fila actual).

✓ **History table (tabla de historial):** con esquema idéntico que contiene versiones anteriores de filas que han sido actualizadas o eliminadas. Cuando una fila en current table se actualiza, SQL Server automáticamente:

a) Establece “SysEndTime” de fila actual al timestamp de transacción

b) Copia fila actualizada a history table

c) Inserta nueva versión en current table con “SysStartTime” establecido a timestamp de transacción y “SysEndTime” a valor máximo

La tabla histórica es append-only por diseño de SQL Server, el sistema previene UPDATE o DELETE directo en la tabla. Solo SQL Server puede escribir mediante mecanismo de system-versioning. Esto proporciona garantía en la inmutabilidad de datos históricos.

4.11.6 Trade-offs y consideraciones de rendimiento

Una de las ventajas de la arquitectura Append-Only es que tiene escritura en disco optimizado, pues típicamente son más rápidas que “updates in place” porque:

✓ No hay necesidad de buscar un registro existente, leer su contenido actual, modificarlo, y escribirlo de vuelta (read-modify-write cycle).

✓ No requiere coordinar bloqueos entre “readers” y “writers” dado que los readers leen versiones inmutables mientras que los writers añaden nuevas versiones sin interferencia.

También se simplifica la concurrencia, dado que la inmutabilidad elimina muchos de sus problemas:

- ✓ No hay necesidad para un bloqueo de tipo “pessimistic locking” o “readers-writer locks” porque los readers nunca bloquean los writers y viceversa.
- ✓ El control de concurrencia optimista es natural, porque cuando se añade una nueva versión, se verifica que la versión esperada es todavía la actual; si no, la transacción se reintenta.
- ✓ No hay “lost updates”, “dirty reads”, o “non-repeatable reads” porque las versiones son inmutables.

Un enfoque append only ofrece de forma natural un rastro de auditoría completo, ya que el historial de cambios queda preservado sin necesidad de construir mecanismos adicionales de auditoría. Cada modificación genera una nueva versión del registro y conserva información temporal sobre cuándo fue válida, lo que permite reconstruir con precisión la evolución de los datos a lo largo del tiempo.

Este mismo principio simplifica notablemente la recuperación ante fallos (crash recovery), al no existir actualizaciones ni sobre escritura de datos, la recuperación se reduce a un procedimiento directo: no es necesario revertir transacciones parcialmente aplicadas sobre páginas de datos, sino únicamente truncar el log append only hasta el último punto de confirmación conocido y continuar desde ahí.

4.12 Desafíos y Limitaciones

Crecimiento de almacenamiento: El desafío más obvio es que los datos históricos consumen espacio de almacenamiento indefinidamente, por ejemplo, una fila que se actualiza 1000 veces genera 1000 versiones. Para contrarrestar esto existen mitigaciones que incluyen:

- ✓ **Compactación:** Eliminar versiones antiguas que ya no son necesarias según política de retención. Sin embargo, esto viola la estricta inmutabilidad y requiere mecanismos cuidadosos para documentar qué fue compactado y por qué.
- ✓ **Archivo en almacenamiento secundario:** Mover versiones

antiguas a un almacenamiento de bajo costo con tiempos de acceso más lentos, pero costo por gigabyte significativamente menor.

✓ **Compresión:** Las versiones sucesivas de una misma entidad frecuentemente tienen alta similitud. En este escenario se puede almacenar solo diferencias entre versiones, a lo que se conoce como "Delta Encoding", o utilizar algoritmos de compresión como zstd y lz4, que pueden reducir significativamente el espacio.

Complejidad del rendimiento de lectura: Leer un estado actual puede requerir filtrar múltiples versiones para encontrar la más reciente. En LSM-Trees, esto significa consultar múltiples memtables, potencialmente generando latencia. Para esto se tiene las siguientes mitigaciones:

✓ **Bloom filters:** Son estructuras de datos probabilísticas que permiten determinar con certeza que una clave no existe en una SSTable sin necesidad de leerla. Esto reduce significativamente las operaciones de entrada y salida en disco y mejora el rendimiento de las búsquedas.

✓ **Índices optimizados:** Se trata de índices filtrados en bases de datos SQL que solo incluyen las filas vigentes, por ejemplo aquellas donde el campo ValidTo es nulo. Esto permite localizar rápidamente el estado actual de un registro sin recorrer versiones históricas.

✓ **Caching agresivo:** Las versiones actuales de los datos suelen considerarse información de uso frecuente. Por ello se mantienen en memoria o en capas de caché de alta velocidad, como Redis o Memcached, para acelerar el acceso y reducir la carga sobre el almacenamiento persistente.

Query complexity para análisis histórico: Pueden existir queries que necesiten analizar patrones a través de tiempo, tales como: trending analysis, time-series aggregations, pero pueden ser complejos de expresar y optimizar. En este punto, es de gran ayuda el "SQL temporal syntax" junto a un Optimizador de Queries que entienda la semántica temporal para generar planes de ejecución

eficientes.

Incompatibilidad con "derecho al olvido": El GDPR en su artículo 17 establece el derecho de los individuos a solicitar la eliminación de sus datos personales bajo ciertas circunstancias. Siendo así, la arquitectura Strict append-only es filosóficamente incompatible con este requisito. Las soluciones pueden ser:

- ✓ **Encryption at rest con eliminación de claves:** Consiste en cifrar los datos personales utilizando claves criptográficas que pueden destruirse cuando un usuario ejerce su derecho al olvido. Al eliminar la clave, la información queda irrecuperable desde el punto de vista práctico, aunque los datos cifrados sigan existiendo físicamente en el almacenamiento.
- ✓ **Pseudonimización:** Implica sustituir identificadores directos por seudónimos, de modo que los datos no puedan atribuirse a una persona sin información adicional. Al eliminar la relación entre el seudónimo y la identidad real, es posible conservar los datos pseudonimizados para análisis o fines estadísticos sin mantener la identificación del individuo.
- ✓ **Aceptar que ciertos registros de auditoría críticos no soportan borrado total:** En sectores como finanzas y salud, la normativa exige la conservación de registros de auditoría durante períodos prolongados, normalmente entre siete y diez años. En estos casos puede surgir un conflicto directo entre la obligación legal de retención y el derecho al olvido. Este conflicto suele resolverse mediante un análisis jurídico y de riesgo en el que las obligaciones de retención prevalecen, siempre que exista una justificación legal clara y proporcional.

4.13 Casos de Uso Óptimos

Las bases de datos append only son especialmente adecuadas en escenarios donde la inmutabilidad y el historial completo de la información no son una opción, sino un requisito. Los siguientes son algunos casos de uso:

- ✓ **Sistemas de Event Sourcing:** El modelo de datos se basa en una secuencia de eventos inmutables que representan cada cambio de estado del sistema. En este contexto, tecnologías como EventStoreDB, Datomic y Kafka encajan de forma natural, ya que están diseñadas para almacenar eventos de manera acumulativa y ordenada.
- ✓ **Registros de auditoría y cumplimiento normativo:** Cuando las regulaciones exigen conservar un historial inalterable de la actividad del sistema, como ocurre en sistemas financieros, de salud o de gestión de registros gubernamentales, el enfoque append only garantiza trazabilidad y evidencia confiable.
- ✓ **Datos de series temporales:** Métricas, logs y datos de sensores se generan como nuevas observaciones que se agregan continuamente, mientras que los datos históricos nunca se modifican. Por esta razón, bases de datos como InfluxDB, TimescaleDB y AWS Timestream suelen implementar almacenamiento append only como principio básico.
- ✓ **Blockchain y distributed ledgers:** Donde la inmutabilidad y el ordenamiento cronológico son propiedades fundamentales. Aunque blockchain añade un mecanismo adicional de consenso distribuido y hashing criptográfico, en esencia funcionan como bases de datos append only.
- ✓ **Almacenamiento de documentos versionados:** Sistemas de gestión de contenido, wikis y plataformas de edición colaborativa requieren mantener el historial completo de revisiones como parte de su funcionalidad. En este sentido, Git puede entenderse como una base de datos append only orientada al control de versiones de código fuente.

4.14 Implementación de Trazas de Auditoría

No todas las operaciones en un sistema requieren auditoría exhaustiva. La determinación de qué eventos deben registrarse debe basarse en un análisis de riesgos y requisitos de cumplimiento. La metodología propuesta por NIST (National Institute of Standards and

Technology, 2024) en su Special Publication 800-92 sugiere clasificar eventos en categorías:

- ✓ **Eventos críticos de seguridad:** Autenticación, autorización, cambios de permisos, acceso a datos sensibles.
- ✓ **Eventos de modificación de datos:** Creación, actualización y eliminación de registros críticos.
- ✓ **Eventos administrativos:** Cambios de configuración, gestión de usuarios, operaciones privilegiadas.
- ✓ **Eventos de cumplimiento regulatorio:** Operaciones específicamente requeridas por regulaciones aplicables.

4.15 Protección Criptográfica de Registros de Auditoría

Consiste en aplicar mecanismos criptográficos para que los eventos registrados por un sistema sean confiables y a prueba de manipulaciones. La idea es que, una vez que un evento queda registrado, no pueda modificarse, eliminarse o insertarse información falsa sin que ello sea evidente. Para lograrlo se emplean técnicas como funciones hash, encadenamiento de registros, firmas digitales y sellos de tiempo (timestamps), que vinculan cada registro con el anterior y con el momento en que fue generado. De esta forma, los logs conservan su integridad, su secuencia temporal y su valor como evidencia, incluso cuando se almacenan o procesan en entornos que no son completamente confiables.

4.15.1 Hash Chains para Detección de Manipulación

La protección criptográfica de registros de auditoría busca garantizar que cualquier alteración, eliminación o inserción fraudulenta de registros sea detectable incluso si un atacante tiene acceso completo al almacenamiento físico donde residen los registros. La técnica fundamental para lograr esta propiedad es el "hash linking" o "chaining", donde cada registro de auditoría incluye un hash criptográfico del registro anterior, creando una cadena vinculada criptográficamente donde alterar cualquier registro rompe la cadena de manera detectable.

El concepto de "hash chains" fue formalizado por Haber y Stornetta en 1991 en su trabajo sobre Timestamping de Documentos Digitales (Haber & Stornetta, 1991), y posteriormente fue aplicado específicamente a logging seguro por Schneier y Kelsey en su paper de 1998 (Schneier & Kelsey, 1998). La idea es elegantemente simple pero criptográficamente robusta: para cada nuevo registro de auditoría generado, se calcula un hash criptográfico que incluye todo el contenido del registro actual más el hash del registro anterior. Este hash se almacena como parte del registro actual.

Formalmente, si se denota el i-ésimo registro de auditoría como R_i y su hash como H_i, entonces $H_i = \text{Hash}(R_i \,\|\, H_{i-1})$, donde $\|$ denota concatenación y Hash es una función de hash criptográfica como SHA-256 o SHA-3. El primer registro (genesis record) requiere tratamiento especial ya que no hay registro anterior; típicamente H_0 se establece a un valor conocido y públicamente disponible, o se deriva de algún valor aleatorio que es almacenado de manera verificable (Kent & Souppaya, 2006).

La propiedad de seguridad clave de hash chains es: cuando se altera cualquier registro histórico R_j requeriría recalcular no solo H_j sino todos los hashes subsecuentes H_{j+1}, H_{j+2}, ..., H_n hasta el registro más reciente. Si el hash del registro más reciente H_n ha sido publicado o almacenado de manera verificable (por ejemplo, entregado a un servicio de timestamping tercero, o escrito a blockchain), entonces un verificador puede detectar que ocurrió una alteración comparando el H_n almacenado externamente con el H_n que resulta de recalcular la cadena desde el genesis record.

4.15.2 Merkle Trees para Verificación Eficiente

Mientras los hashes chains proporcionan detección fuerte de manipulación, tienen una limitación práctica: verificar la integridad de la cadena completa requiere procesar todos los registros desde el origen (genesis record) hasta el registro más reciente. Esto conlleva que, para registros de auditoría con millones o miles de millones de registros, esta verificación completa puede ser computacionalmente prohibitiva. Entonces los Merkle trees (también llamados hash

trees) (Merkle, 1979) proporcionan una alternativa que permite la verificación eficiente de subconjuntos de registros sin procesar todo el árbol.

Un Merkle tree es una estructura de árbol binario donde cada nodo hoja contiene el hash de un registro de datos, y cada nodo interno contiene el hash de la concatenación de sus dos nodos hijos. El nodo raíz en la cima del árbol (el Merkle root) y representa criptográficamente todos los registros en el árbol. Ralph Merkle propuso esta estructura en 1979 originalmente para firmas digitales eficientes, pero ha encontrado aplicaciones amplias incluyendo Bitcoin, Git, y sistemas de archivos distribuidos (Merkle, 1979).

Para un conjunto de n registros de auditoría, éstos se colocan en las n hojas del árbol. Se calculan hashes para cada hoja, luego hashes para cada nivel de nodos internos basándose en sus hijos, hasta llegar a la raíz (root). El Merkle root puede entonces ser publicado o almacenado de manera verificable.

4.15.3 Firmas Digitales y No Repudio

Los Hash Chains y Merkle Trees protegen contra la alteración de registros, pero no proporcionan el efecto No Repudio, esto es la propiedad de que el creador de un registro no puede posteriormente negar lo creado. Para no repudio, se requieren firmas digitales basadas en criptografía de clave pública. Una firma digital sobre un registro de auditoría proporciona evidencia criptográfica de que el registro fue generado por una entidad específica (el poseedor de la clave privada correspondiente a la clave pública verificadora) y no ha sido alterado desde su firma.

El proceso de firma típicamente involucra calcular un hash del registro, encriptar ese hash con la clave privada del firmante, y adjuntar la firma resultante al registro. Un verificador puede luego desencriptar la firma usando la clave pública del firmante, calcular el hash del registro, y comparar los dos. Si coinciden, el verificador tiene evidencia criptográfica que: (a) el registro no ha sido alterado desde que fue firmado, y (b) fue firmado por el poseedor de la clave privada correspondiente a la clave pública usada para verificación.

Para registros de auditoría, las firmas digitales pueden aplicarse a registros individuales o a lotes de registros. Firmar cada registro individual proporciona máxima granularidad de no repudio, pero tiene alta sobrecarga computacional (overhead). Firmar lotes de registros reduce el overhead pero significa que el no repudio aplica al lote completo, no a registros individuales.

4.15.4 Timestamping Confiable (RFC 3161)

Para muchos propósitos de auditoría, es crítico no solo registrar qué ocurrió sino cuándo ocurrió con certeza verificable. Sin embargo, timestamps generados localmente por el sistema que genera registros de auditoría son inherentemente no confiables, un atacante con control del sistema podría retroactivamente alterar el reloj del sistema y regenerar registros con timestamps fraudulentos. Trusted timestamping (sellado de tiempo confiable) resuelve este problema mediante intervención de una tercera parte confiable.

El Internet Engineering Task Force (IETF) estandarizó un protocolo para trusted timestamping en RFC 3161, publicado en 2001 (Adams et al., 2001). El protocolo define como un cliente puede enviar un hash de un documento (o en el presente caso, un registro de auditoría) a una Time Stamping Authority (TSA) confiable, y recibir de vuelta un time stamp token que certifica que el documento existía en el momento especificado por la TSA.

El time stamp token contiene: el hash del documento conteniendo timestam, el tiempo preciso según el reloj de la TSA, un número serial único, y una firma digital de la TSA sobre toda esta información. Relevantemente, la TSA mantiene un registro de auditoría propio de todos los timestamp tokens que ha emitido, proporcionando evidencia verificable de cuándo fueron emitidos. Si posteriormente hay disputa sobre cuándo se creó un registro de auditoría, el timestamp token puede ser verificado criptográficamente y proporciona evidencia convincente.

4.16 Análisis forense de registros de auditoría

4.16.1 Metodología de Investigación Forense Digital

El análisis forense de registros de auditoría, la práctica de examinar registros de auditoría para reconstruir eventos, identificar causas de incidentes, y desarrollar evidencia para propósitos legales o administrativos, requiere metodología rigurosa para asegurar que los hallazgos sean defendibles. El campo de la forense digital ha desarrollado frameworks metodológicos que, aunque diseñados originalmente para análisis de sistemas de archivos y memoria, se aplican igualmente a análisis de logs.

El framework más ampliamente citado es el propuesto por la DFRWS (Palmer, 2001) que describe un modelo de cuatro fases: colección, examen, análisis y reporte. La fase de colección involucra identificar, adquirir y preservar fuentes de evidencia potencial, en el presente caso, registros de auditoría relevantes. En esta fase es importante mantener la cadena de custodia y asegurar la integridad de evidencia mediante hashing criptográfico y documentación meticulosa de cómo fueron obtenidos los logs.

La fase de examen involucra procesar los datos recolectados para extraer información relevante. Para registros de auditoría, esto puede involucrar "parsing" de logs en formatos estructurados, filtering para remover "noise", enriquecimiento con datos contextuales de otras fuentes, y organización temporal de eventos. La fase de análisis es donde ocurre el trabajo investigativo real, desarrollando y probando hipótesis sobre qué ocurrió, correlacionando eventos de múltiples fuentes, identificando patrones anómalos, reconstruyendo secuencias de actividad.

4.16.2 Reconstrucción de Líneas de Tiempo de Eventos

Una técnica fundamental en análisis forense de logs es la construcción de líneas de tiempo, representaciones cronológicas de eventos relevantes que permiten a investigadores visualizar y razonar sobre secuencias de actividad. Las líneas de tiempo son particularmente valiosas cuando se investigan incidentes complejos que involucran

múltiples actores, múltiples sistemas, y períodos extendidos de tiempo.

Construir una línea de tiempo precisa de registros de auditoría presenta varios desafíos técnicos. Primero está el problema de sincronización de tiempo. Eventos en la línea de tiempo provienen típicamente de múltiples sistemas que pueden tener relojes desincronizados. Incluso con Network Time Protocol (NTP), la sincronización de relojes típicamente tiene precisión solo en el rango de milisegundos a segundos. Para eventos que ocurren en rápida sucesión o casi simultáneamente en diferentes sistemas, determinar orden temporal verdadero puede ser imposible basándose solo en timestamps físicos.

4.16.3 Detección de Alteración de Logs

Un desafío particular en análisis forense es determinar si los mismos registros de auditoría han sido manipulados. Un atacante sofisticado que ha ganado acceso suficiente a un sistema puede intentar cubrir sus huellas alterando o eliminando registros de auditoría que documentarían su actividad. Detectar tal manipulación requiere tanto mecanismos preventivos (las protecciones criptográficas discutidas anteriormente) como técnicas analíticas para identificar indicadores de manipulación.

La técnica más robusta es la verificación criptográfica mediante hash chains o Merkle trees como se discutió previamente. Si los registros de auditoría fueron protegidos mediante estas técnicas y los hashes de verificación fueron almacenados de manera externa, entonces la alteración es algorítmicamente detectable. Sin embargo, en muchas organizaciones, especialmente aquellas con madurez de seguridad limitada, los registros de auditoría no tienen protección criptográfica. En estos casos, los analistas deben confiar en indicadores heurísticos de manipulación.

Capítulo 5

ANÁLISIS Y GENERACIÓN DE REPORTES DE AUDITORÍA

5.1 Análisis de Datos de Auditoría

Los sistemas de auditoría modernos generan volúmenes masivos de datos, pueden producir gigabytes o terabytes de eventos de registro diariamente. Este volumen presenta un desafío fundamental: cómo transformar cantidades masivas de eventos de registro crudos en información accionable que permita detectar amenazas, identificar problemas operacionales, demostrar cumplimiento regulatorio, y soportar investigaciones forenses.

El Instituto Nacional de Estándares y Tecnología (NIST) (National Institute of Standards and Technology, 2024) en su Publicación Especial 800-92 "Guía para la Gestión de Registros de Seguridad Informática" establece que "el análisis efectivo de registros requiere reducir grandes volúmenes de datos de registro a conjuntos más pequeños de información relevante mediante procesos automatizados de filtrado, normalización, agregación y correlación". Sin estos procesos, las pistas de auditoría se vuelven inútiles en la práctica, la información existe técnicamente pero no puede ser extraída eficientemente cuando se necesita.

El análisis de datos de auditoría típicamente sigue una cadena de múltiples etapas que transforma progresivamente datos crudos en inteligencia accionable:

5.1.1 Etapa 1: Recolección y Agregación

La primera etapa consiste en recolectar eventos de registro de fuentes distribuidas y centralizarlos en repositorio común. Como se discutió en el patrón Log Aggregation, esto involucra agentes de recolección (Filebeat, Fluentd, Logstash) que leen registros locales y los transmiten a un agregador central. La agregación debe preservar metadatos críticos: marca temporal (timestamp) original del evento (no solo marca temporal de recolección), fuente que generó el evento (nombre de servidor, dirección IP, servicio, componente), severidad o criticidad del evento, y contexto adicional específico de dominio.

El desafío de agregación en sistemas distribuidos es la sincronización temporal, diferentes servidores pueden tener relojes con

desviación de segundos o minutos. El Protocolo de Tiempo de Red (NTP, Network Time Protocol) debe configurarse en todos los nodos para mantener sincronización temporal dentro de rangos aceptables (típicamente menos de 1 segundo de desviación). NIST SP 800-92 recomienda que "todas las fuentes de registro en una organización deben utilizar misma fuente de tiempo de referencia y estar configuradas para ajustar automáticamente sus relojes según sea necesario para mantenerse sincronizados" (National Institute of Standards and Technology, 2024).

5.2 Etapa 2: Normalización y Análisis Sintáctico (Parsing)

Los eventos de registro Log llegan en formatos heterogéneos, syslog, JSON estructurado, XML, texto plano sin estructura, formatos propietarios de aplicaciones específicas. La normalización convierte estos formatos diversos a esquema común que permite procesamiento uniforme.

El análisis sintáctico extrae campos estructurados de texto no estructurado. Por ejemplo, una línea de registro del servidor web Apache:

```
192.168.1.100 - usuario123 [09/Ene
   /2026:14:30:45 +0000] "GET /api/ordenes/123
    HTTP/1.1" 200 1234
```

Debe analizarse para extraer campos: dirección IP del cliente (192 .168.1.100), usuario autenticado (usuario123), marca temporal (09/E ne/2026:14:30:45), método HTTP (GET), ruta solicitada (/api/orde nes/123), código de estado (200), bytes transferidos (1234).

5.3 Etapa 3: Enriquecimiento

El enriquecimiento añade contexto adicional a eventos mediante consulta en fuentes de datos auxiliares. Ejemplos de enriquecimiento incluyen:

✓ **Resolución de identidad:** El identificador numérico de usuario

o GUID se enriquece consultando por ejemplo un directorio corporativo (Active Directory, LDAP) para añadirle nombre completo, correo electrónico, departamento, supervisor, roles organizacionales, etc. Esto convierte "IDUsuario: 12345" en información más útil para analistas: "Juan Pérez, Departamento Financiero, rol: Contador Senior".

- ✓ **Geolocalización de direcciones IP:** Las direcciones IP públicas se enriquecen mediante bases de datos de geolocalización (Ejemplo: MaxMind GeoIP2, IP2Location) para añadir país, ciudad, coordenadas geográficas, número de sistema autónomo, tipo de conexión. Esto permite detectar anomalías geográficas, por ejemplo, cuando un usuario normalmente conectándose desde Ecuador súbitamente tiene inicio de sesión desde Rusia.
- ✓ **Fuentes de inteligencia de amenazas:** Se utilizan para detectar si un evento de seguridad está relacionado con actividades maliciosas ya conocidas. Para ello, se contrastan indicadores técnicos tales como: direcciones IP, nombres de dominio o hashes de archivos, contra repositorios especializados de inteligencia de amenazas, entre ellos AlienVault OTX, Abuse.ch y MISP. Cuando alguno de estos indicadores coincide con registros previamente asociados a actividades maliciosas, por ejemplo, una dirección IP identificada como servidor de comando y control de una botnet, el evento se clasifica automáticamente como de alta prioridad, ya que existe evidencia concreta de compromiso.
- ✓ **Inventario de activos:** Permite dar contexto a los eventos de seguridad al enriquecer información básica como el nombre del servidor o la dirección IP mediante consultas a la Base de Datos de Gestión de Configuración. A partir de estas consultas se incorporan datos relevantes del activo, como su nivel de criticidad, el responsable del sistema, la versión del software y la clasificación de la información que procesa. Este contexto hace posible priorizar adecuadamente los incidentes. Por ejemplo, un evento que afecta a un sistema que procesa datos de tarjetas de pago bajo el estándar PCI DSS resulta más crítico que el mismo evento ocurrido en un entorno de desarrollo.

✓ **Detalles de transacciones de negocio:** Identificadores de transacciones se enriquecen consultando sistemas operacionales para añadir detalles de negocio. Por ejemplo, "IDOrden: 98765" se enriquece con monto de orden, cliente, productos, estado, permitiendo analizar patrones a nivel de negocio en lugar de solo nivel técnico.

Tomar en cuenta que el enriquecimiento debe balancearse contra latencia, pues consultar múltiples sistemas externos por cada evento puede introducir retrasos significativos. Siendo así, las estrategias de optimización incluyen almacenamiento en caché de consultas frecuentes, agrupación de múltiples consultas, y enriquecimiento asíncrono donde un contexto adicional se añade después de una ingesta inicial sin bloquear procesamiento.

5.4 Etapa 4: Filtrado y Reducción

No todos los eventos tienen igual valor para análisis. El filtrado elimina eventos de bajo valor o ruido, reduciendo volumen de datos que debe almacenarse y analizarse. NIST SP 800-92 identifica varios tipos de filtrado:

✓ **Filtrado por prioridad:** Retener solo eventos que exceden un umbral de severidad. Por ejemplo, retener solo eventos ADVERTENCIA, ERROR, CRÍTICO; descartar eventos DEPURACIÓN e INFORMACIÓN para sistemas no críticos. Esto reduce el volumen, pero mantiene información sobre eventos potencialmente problemáticos.

✓ **Filtrado por relevancia de seguridad:** En un contexto de monitoreo de seguridad, filtrar para retener solo eventos con implicaciones de seguridad. Por ejemplo, retener todos los eventos de autenticación (éxitos y fallos), acceso a recursos sensibles, cambios de configuración, elevación de privilegios; descartar eventos operacionales rutinarios sin implicaciones de seguridad.

✓ **Eliminación de duplicados:** Cuando un mismo evento se genera repetidamente (por ejemplo, servidor intentando conectar a un

recurso no disponible genera error cada segundo), la eliminación de duplicados se puede contraer en una entrada única con contador de cuántas veces ocurrió. Esto preserva información de que problema ocurrió sin almacenar millones de copias del mismo mensaje.

✓ **Muestreo:** Para sistemas con volúmenes extremadamente altos, el muestreo estadístico retiene solo un porcentaje de eventos (por ejemplo, 1 de cada 100 solicitudes HTTP) mientras mantiene representatividad estadística de patrones. El muestreo reduce volumen dramáticamente, pero pierde granularidad para investigaciones forenses específicas.

El filtrado debe aplicarse cuidadosamente, filtrado excesivamente agresivo puede eliminar evidencia crítica. Por ejemplo, el PCI DSS Requisito 10.5 (PCI Security Standards Council, 2022) específicamente prohíbe la alteración o eliminación de registros de auditoría sin mecanismos apropiados de protección y retención. Las decisiones de filtrado deben documentarse en política de registro y revisarse periódicamente.

5.5 Etapa 5: Correlación y Agregación

La correlación combina eventos relacionados de múltiples fuentes para identificar patrones complejos que no serían visibles examinando eventos individuales aisladamente. Existen múltiples tipos de correlación:

✓ **Correlación temporal:** Eventos que ocurren dentro de una ventana de tiempo específica se pueden agrupar. Por ejemplo, inicio de sesión exitoso seguido dentro de 2 segundos por acceso a archivo sensible, seguido dentro de 5 segundos por transferencia de red grande puede indicar exfiltración de datos.

✓ **Correlación por entidad:** Todos los eventos relacionados con una misma entidad (usuario, dirección IP, sesión, transacción) se agrupan. Esto permite rastrear actividad completa de la entidad a través del tiempo. Por ejemplo, agrupar todos los eventos de usuario específico durante su sesión de trabajo permite detectar

anomalías en patrón de comportamiento del usuario.

- ✓ **Correlación entre fuentes:** Eventos de diferentes sistemas se correlacionan cuando representan diferentes perspectivas de una misma actividad. Por ejemplo: registro de firewalls mostrando conexión permitida, registro de servidor web mostrando solicitud HTTP, registro de aplicación mostrando consulta de base de datos, registro de auditoría de base de datos mostrando ejecución de consulta. Correlacionar estos eventos proporciona una vista completa de extremo a extremo de la transacción.
- ✓ **Correlación mediante reglas:** Reglas predefinidas especifican patrones de eventos que indican condiciones de interés. Por ejemplo, regla de detección de ataque de fuerza bruta: "Si se observan 10 o más intentos de inicio de sesión fallidos desde una misma dirección IP para una misma cuenta dentro de 5 minutos, se debe generar una alerta de severidad alta". Los sistemas de Gestión de Información y Eventos de Seguridad (SIEM) implementan motores de correlación basados en reglas que evalúan miles de reglas contra flujos de eventos en tiempo real.

5.6 Técnicas de Análisis: De Descriptivo a Predictivo

El análisis de datos de auditoría abarca un espectro desde el análisis descriptivo retrospectivo hasta el análisis predictivo prospectivo:

5.6.1 Análisis Descriptivo: ¿Qué Ocurrió?

El análisis descriptivo examina eventos históricos para entender qué ocurrió en el tiempo. Se incluyen las siguientes técnicas:

- ✓ **Tableros y visualizaciones:** Son representaciones gráficas de métricas clave que proporcionan una vista rápida del estado del sistema. Los tableros típicamente incluyen: gráficos de líneas mostrando volumen de eventos a través del tiempo, gráficos de barra mostrando distribución de eventos por categoría o fuente, mapas de calor mostrando actividad por hora del día y día de semana, indicadores mostrando métricas clave contra umbrales, y tablas mostrando los N usuarios, direcciones IP, o recursos más

activos.

Al respecto, las herramientas de software: Kibana, Grafana, y Splunk, proporcionan capacidades ricas de creación de tableros. El diseño efectivo de tableros sigue principios de visualización estandarizados, maximizan la proporción de elementos dedicados a mostrar datos versus decoración, evitan elementos decorativos que no añaden información, y usan codificaciones visuales apropiadas para tipos de datos (posición para valores cuantitativos precisos, color para categorías, tamaño de área para magnitudes relativas).

✓ **Consultas específicas:** Búsquedas específicas para responder preguntas puntuales. "¿Cuántas veces el usuario X accedió al sistema la semana pasada?" "¿Qué archivos fueron modificados en el servidor Y el 5 de enero?". Para el efecto, la herramienta Elasticsearch proporciona lenguaje de consulta e interfaz visual (Kibana) para consultas específicas. Splunk, proporciona un Lenguaje de Procesamiento de Búsqueda (SPL) para expresar consultas complejas.

✓ **Informes programados:** Informes que se generan automáticamente en cadencia regular (diariamente, semanalmente, mensualmente) y se distribuyen a los interesados. Los informes de cumplimiento típicamente documentan métricas requeridas por regulaciones, ejemplo para PCI DSS: número de intentos de acceso a datos de tarjetas, cambios a configuración de sistemas en alcance, accesos de cuentas privilegiadas. Para HIPAA: accesos a Información de Salud Protegida, divulgación a terceros, accesos de emergencia.

5.6.2 Análisis Diagnóstico: ¿Por Qué Ocurrió?

El análisis diagnóstico investiga causas raíz de eventos o patrones observados. Sus técnicas incluyen:

✓ **Análisis de profundización:** Partir de una métrica agregada y progresivamente descomponer en componentes más granulares para identificar fuente de anomalía. Por ejemplo, un tablero muestra que la tasa de error global de interfaz de programación aumentó 500 %, la profundización por punto de acceso revela

específicamente que la interfaz "/api/pagos" tiene 95 % de errores. La profundización por código de error revela que todos son 503 Servicio No Disponible. La profundización por marca temporal revela que comenzó exactamente a las 14:30. La correlación con registros de despliegue revela que nueva versión de servicio de pagos fue desplegada a las 14:28. Conclusión: nuevo despliegue introdujo error que causa indisponibilidad de servicio.

- ✓ **Análisis de causa raíz mediante correlación:** Cuando un incidente complejo ocurre, se debe correlacionar eventos de múltiples sistemas para reconstruir una cadena causal. Por ejemplo, la interrupción de una aplicación web puede rastrearse hacia atrás: registros de aplicación muestran tiempos de espera conectando a base de datos, los registros de base de datos muestran consultas bloqueadas esperando bloqueos, las métricas de rendimiento de base de datos muestran pico en espera de entrada/salida, los registros de arreglo de almacenamiento muestran fallos de disco, las alertas de monitoreo muestran que modo degradado de RAID comenzó 10 minutos antes de los primeros síntomas visibles a usuarios.
- ✓ **Análisis comparativo:** Comparar períodos diferentes para identificar qué cambió. Por ejemplo, la latencia de interfaz de programación aumentó esta semana comparada con la semana pasada. El análisis comparativo revela que el volumen de solicitudes aumentó 40 %, pero eso solo explica una parte del aumento de latencia. Entonces se puede ampliar y descubrir que la distribución de tipos de solicitudes cambió, la proporción de solicitudes de tipo "consulta_compleja" aumentó de 10 % a 30 %. La investigación podría revelar que la nueva funcionalidad lanzada esa semana generó más consultas complejas.

5.6.3 Análisis Predictivo: ¿Qué Ocurrirá?

El análisis predictivo utiliza técnicas de aprendizaje automático y estadística para predecir eventos futuros basándose en patrones históricos; así se tiene:

Detección de anomalías: Los algoritmos de detección de

anomalías aprenden patrones normales de comportamiento de datos históricos y generan alertas cuando observan desviaciones significativas. Técnicas incluyen:

- ✓ **Detección estadística de anomalías:** Modela distribución estadística de métrica (por ejemplo, número de inicios de sesión por hora) y detecta valores que están más de N desviaciones estándar de la media.
- ✓ **Pronóstico de series temporales:** Modelos como ARIMA (Media Móvil Integrada AutoRegresiva), Prophet (desarrollado por Facebook), o redes neuronales LSTM (Memoria de Corto y Largo Plazo) predicen un valor esperado de serie temporal en una marca temporal futura basándose en histórico. Las anomalías son valores observados que difieren significativamente de la predicción.
- ✓ **Detección de anomalías basada en agrupamiento:** Algoritmos como DBSCAN (Agrupamiento Espacial Basado en Densidad de Aplicaciones con Ruido) o Bosque de Aislamiento agrupan eventos similares en grupos. Los eventos que no pertenecen a ningún grupo denso (valores atípicos) se marcan como anomalías.

Ya se ha mencionado antes sobre la herramienta de software Elasticsearch, ésta incluye características de aprendizaje automático que automáticamente detectan anomalías en series temporales. El Kit de Herramientas de Aprendizaje Automático de Splunk proporciona algoritmos para análisis de registros. Hay otros productos especializados como Anodot, que se enfocan exclusivamente en detección automática de anomalías en métricas de negocio y sistemas.

Análisis de Comportamiento de Usuarios y Entidades (UEBA): UEBA aplica aprendizaje automático para construir perfiles de comportamiento normal de usuarios y entidades (servidores, aplicaciones, cuentas de servicio) y detectar desviaciones que pueden indicar compromiso de cuenta o amenaza interna. Por ejemplo, UEBA puede detectar:

- ✓ Usuario que normalmente trabaja de 9am-5pm súbitamente tiene actividad a las 3am.

✓ Usuario que normalmente accede a 10-20 archivos por día súbitamente accede a 5,000 archivos.

✓ Usuario que normalmente se conecta desde Ecuador súbitamente tiene inicio de sesión desde China.

✓ Cuenta de servicio que normalmente ejecuta consultas SELECT súbitamente ejecuta DROP TABLE.

Mantenimiento predictivo y planificación de capacidad: El análisis de tendencias en métricas operacionales predice cuándo los recursos alcanzarán una capacidad o cuándo sus componentes probablemente fallarán. Por ejemplo, el análisis de crecimiento de uso de disco predice qué servidor alcanzará 90 % de capacidad en 3 semanas, permitiendo aprovisionar almacenamiento adicional proactivamente. El análisis de tasa de errores en un arreglo de almacenamiento predice probabilidad de fallo de disco en los próximos 30 días (basándose en métricas SMART), permitiendo un reemplazo proactivo antes de la pérdida de datos.

5.7 Plataformas Integradas

Los sistemas denominados SIEM, son módulos integrados para la Gestión de Información y Eventos de Seguridad, éstos integran funcionalidad de recolección, normalización, correlación, análisis, y generación de alertas en plataforma unificada. Una arquitectura SIEM típica consta de los siguientes componentes:

✓ **Capa de recolección de datos:** Conectores para múltiples fuentes de registro (syslog, Registro de Eventos de Windows, interfaces de programación de proveedores de nube, agentes instalados en puntos finales, datos de flujo de red de cortafuegos y enrutadores, registros de aplicaciones).

✓ **Motor de normalización:** Analizador que convierte formatos heterogéneos a un modelo de datos común (CEF, LEEF, esquema propietario del proveedor).

✓ **Motor de correlación:** Motor de reglas que evalúa eventos contra

una biblioteca de reglas de correlación para detectar patrones que indican incidentes de seguridad. Las reglas típicamente se expresan como: SI (condición_1 Y condición_2 Y ...) DENTRO_DE (ventana_temporal) ENTONCES (generar_alerta).

- ✓ **Generación de alertas y gestión de casos:** Sistema de gestión de alertas que prioriza alertas basándose en su severidad, agrupa alertas relacionadas en casos (incidentes), asigna casos a analistas, y rastrea estado de investigación y remediación.
- ✓ **Informes de cumplimiento:** Informes predefinidos mapeados a requisitos de regulaciones comunes (PCI DSS, HIPAA, SOX, GDPR, ISO 27001) que documentan cumplimiento con controles específicos.

Los proveedores principales de SIEM incluyen a: Splunk Enterprise Security, IBM QRadar, LogRhythm, ArcSight (Micro Focus), Microsoft Azure Sentinel (SIEM nativo de nube), Google Chronicle (SIEM nativo de nube), y soluciones de código abierto como OSSEC, AlienVault OSSIM, y Wazuh.

Capítulo 6

CASOS DE USO VERTICALES

6.1 Sector Financiero y Fintech

El sector financiero se mueve dentro de algunos de los marcos regulatorios más exigentes del mundo en lo que respecta a auditoría y trazabilidad. En Estados Unidos, distintos reguladores tales como la Federal Reserve (Sistema de la Reserva Federal), la Office of the Comptroller of the Currency (Oficina del Contralor de la Moneda), la Securities and Exchange Commission, SEC (Comisión de Bolsa y Valores), la Financial Industry Regulatory Authority, FINRA (Autoridad Reguladora de la Industria Financiera) y la Commodity Futures Trading Commission, CFTC (Comisión de Comercio de Futuros de Productos Básicos) entre otros, establecen obligaciones estrictas de conservación de registros y auditoría. Estas exigencias no se limitan a la protección del consumidor: también responden a la necesidad de preservar la estabilidad del sistema financiero en su conjunto y de combatir de forma efectiva el crimen financiero.

Un área particular de enfoque es la auditoría de sistemas de trading. Las regulaciones MiFID II en Europa y Reg NMS en Estados Unidos imponen requisitos estrictos de "order audit trail", la capacidad de rastrear cada orden desde su creación hasta la ejecución, capturando todas las modificaciones, cancelaciones, y coincidencias. Los sistemas de trading deben capturar timestamps con precisión de microsegundos, permitiendo a reguladores reconstruir exactamente qué ocurrió durante condiciones volátiles de mercado.

Otro requisito crítico es auditoría para prevención de lavado de dinero (AML - Anti-Money Laundering). El Bank Secrecy Act y regulaciones subsecuentes requieren que las instituciones financieras mantengan programas completos de AML que incluyen monitoreo de transacciones para patrones sospechosos. Los sistemas de auditoría deben capturar suficiente información sobre cada transacción para permitir análisis de patrones, originador, beneficiario, monto, frecuencia, relación con otras transacciones, y cualquier flag de riesgo identificado por sistemas automáticos.

6.2 Sector Salud: Registros Médicos Electrónicos

Los sistemas de registros médicos electrónicos (EHR - Electronic Health Records) presentan requisitos particularmente desafiantes para auditoría debido a la sensibilidad extrema de los datos de salud, el número grande de usuarios con acceso legítimo (médicos, enfermeras, técnicos, personal administrativo), y escenarios complejos de acceso como emergencias donde el acceso normal basado en roles puede necesitar ser de característica "override".

La HIPAA Security Rule requiere específicamente que las entidades cubiertas implementen "hardware, software, y/o mecanismos procedurales que registren y examinen actividad en sistemas de información que contienen o usan electronic protected health information". Los registros de auditoría de EHR típicamente deben capturar: quién accedió al registro de qué paciente, cuándo, desde dónde, qué información específicamente fue vista o modificada, e idealmente el propósito clínico o razón de negocio para el acceso.

Un escenario particularmente complejo es el "break-glass Access", donde en situaciones de emergencia el personal clínico necesita acceso inmediato a un registro del paciente sin tiempo para obtener autorizaciones normales. Los sistemas EHR auditables deben soportar este acceso de emergencia mientras capturan cuidadosamente cada instancia para revisión posterior. El Rastro de Auditoría debe registrar que ocurrió un break-glass access, quién lo invocó, para qué paciente, en qué circunstancias, y debe alertar a personal encargado de una revisión retrospectiva que verifica que el acceso fue apropiado.

6.3 Facturación Electrónica en Ecuador: Sistema SRI

El Servicio de Rentas Internas (SRI) del Ecuador implementó un sistema obligatorio de facturación electrónica que presenta requisitos específicos de auditoría y trazabilidad. Este sistema, vigente desde 2015 y mandatorio para virtualmente todos los contribuyentes desde 2018, representa uno de los regímenes de facturación electrónica más completos en América Latina y proporciona un caso de estudio valioso sobre implementación de auditabilidad en contextos de gobierno.

6.3.1 Arquitectura del Sistema de Facturación Electrónica

El sistema de facturación electrónica del SRI está basado en varios componentes técnicos clave. Primero, las facturas electrónicas deben generarse en formato XML siguiendo esquemas XSD específicos publicados por el SRI, que definen la estructura exacta de diferentes tipos de comprobantes (facturas, notas de crédito, notas de débito, guías de remisión, comprobantes de retención). Estos esquemas son extremadamente detallados, especificando cada campo que debe estar presente, formatos de datos, validaciones, y códigos válidos para clasificación de productos y servicios.

Segundo, cada comprobante electrónico debe ser firmado digitalmente usando el estándar XAdES-BES (XML Advanced Electronic Signatures, Basic Electronic Signature). XAdES-BES es un formato de firma digital que extiende XML Signature para proporcionar propiedades adicionales requeridas para propósitos legales en Europa y adoptados en Ecuador. La firma debe ser generada usando un certificado digital emitido por una entidad certificadora autorizada por el SRI, típicamente obtenido mediante un proceso de verificación de identidad del contribuyente.

6.3.2 Implementación de Firma Digital XAdES-BES

La implementación de firma XAdES-BES para facturación electrónica requiere comprensión tanto de los estándares XML Signature como de las extensiones específicas de XAdES. En .NET, esto típicamente se implementa usando la clase SignedXml del namespace System.Security.Cryptography.Xml junto con bibliotecas especializadas para XAdES como FirmaXadesNet o Xades.Net.

El proceso de firma comienza con el XML del comprobante electrónico completo. Este XML debe primero ser canonicalizado usando algoritmo de canonicalización especificado (típicamente Canonical XML 1.0), calculándose su digest usando SHA-256 o SHA-1. El digest se incluye en un elemento Reference dentro de la estructura de firma, junto con información sobre el algoritmo usado.

La estructura de firma XAdES incluye varios elementos específicos. El elemento SignedProperties contiene propiedades firmadas como

el timestamp de firma y el certificado del firmante. El elemento SignedSignatureProperties contiene metadata sobre la firma misma. El elemento SignedDataObjectProperties puede contener metadata sobre el objeto siendo firmado. Todos estos elementos están incluidos en el cálculo de la firma, asegurando que no pueden ser alterados sin invalidar la firma.

6.3.3 Proceso de Autorización y Rastro de Auditoría

Una vez que un comprobante electrónico ha sido generado y firmado, debe ser enviado al SRI para su autorización. Este proceso ocurre en tiempo real o en modo offline, y genera un registro de auditoría completo de todo el ciclo de vida del comprobante. El contribuyente emisor envía el XML firmado al servicio web del SRI mediante SOAP o REST API.

El SRI realiza múltiples validaciones: esquema XML contra el XSD oficial, firma digital, validez del certificado digital, secuencialidad de números de comprobante, existencia y estado del RUC del emisor y receptor, códigos de productos y servicios, cálculo de valores (subtotales, IVA, total), y más. Si todas las validaciones pasan, el SRI genera una clave de acceso única de 49 dígitos que identifica el comprobante y lo marca como AUTORIZADO.

El Rastro de Auditoría completo de este proceso debe incluir: timestamp de generación del comprobante, timestamp de firma digital, hash del XML firmado, timestamp de envío al SRI, respuesta del SRI (autorizado, rechazado, devuelto), clave de acceso si fue autorizado, cualquier mensaje de error si fue rechazado, timestamp de emisión del RIDE (Representación Impresa del Documento Electrónico), y registros de cualquier anulación posterior o nota de crédito/débito relacionado.

6.4 BANRED Ecuador: Switch Interbancario y Compliance PCI-DSS

BANRED S.A., fundada en 1984, es la red interbancaria más grande de Ecuador. Opera más de 7,400 cajeros automáticos interconectados, procesa aproximadamente 1,200 millones de transacciones

anuales (proyección 2024) y conecta a 44 clientes directos y más de 100 clientes indirectos. BANRED mantiene certificación PCI-DSS (Payment Card Industry Data Security Standard) desde 2014, recertificándose anualmente. Entre los años 2022 y 2023, renovó completamente su infraestructura tecnológica actualizando su core transaccional y switch para mejorar la velocidad y seguridad. Se toma este caso dado su complejidad tecnológica y cumplimiento estricto de estándares internacionales respecto de su infraestructura, operatividad, seguridad y auditabilidad.

Respecto de los desafíos de auditabilidad, BANRED no solamente involucra Compliance PCI-DSS, además mantiene una interoperabilidad multibanco y un volumen transaccional masivo de varios servicios. Detallando, como switch interbancario, conecta instituciones financieras diversas (bancos privados, cooperativas de ahorro y crédito, mutualistas), cada transacción involucra al menos dos instituciones: emisora y adquirente. Los registros de auditoría deben permitir rastrear transacción completa desde el origen hasta el destino, identificando responsabilidades en caso de disputas. Procesa transacciones que involucran datos de tarjetas de débito y crédito de múltiples emisores, PCI-DSS Requisito 10 exige registros de auditoría completos de todo acceso a datos de tarjetas, operaciones administrativas en sistemas de procesamiento, y acceso físico a centros de datos para 1,200 millones de transacciones anuales.

BANRED procesa aproximadamente 38 transacciones por segundo en promedio, el sistema de auditoría debe capturar eventos a esta escala sin degradar performance transaccional al operar múltiples servicios: red de cajeros automáticos, Pago Directo (transferencias interbancarias en tiempo real), WIP (plataforma de pagos móviles lanzada 2024), red de cobros y pagos para entidades públicas/privadas, y gestión de pagos del Bono de Desarrollo Humano. BANRED además opera cámara de compensación para liquidar transacciones entre instituciones participantes. Cada servicio tiene requisitos específicos de auditoría.

Adicional, la Superintendencia de Bancos y Seguros, así también la Superintendencia de Economía Popular y Solidaria, son instituciones

reguladoras de las operaciones financieras en Ecuador que supervisan a las instituciones financieras conectadas a BANRED, por tanto, su capacidad de proporcionar registros de auditoría detallados facilita auditorías regulatorias de bancos participantes. Durante una auditoría, un banco puede necesitar demostrar que sus transacciones procesadas a través de BANRED fueron manejadas apropiadamente, entonces BANRED proporciona evidencia mediante logs.

La certificación PCI DSS de BANRED constituye un requisito esencial para que los bancos ecuatorianos puedan aceptar y procesar pagos con tarjetas de redes internacionales como Visa, Mastercard y American Express. Esta certificación garantiza que la infraestructura que soporta las transacciones cumple con estándares estrictos de seguridad y auditoría. En caso de que BANRED pierda dicha certificación, las redes internacionales podrían suspender su conexión, lo que tendría como consecuencia directa la imposibilidad de utilizar tarjetas de pago en el país. Este escenario afectaría tanto a usuarios como a entidades financieras. Por ello, el cumplimiento riguroso de los requisitos de auditoría por parte de BANRED resulta crítico para la operación continua y la estabilidad del sistema financiero nacional.

Capítulo 7

IMPLEMENTACIÓN PRÁCTICA EN ENTORNOS .NET

7.1 Diseño de Arquitectura para un Software Auditable

La implementación de software auditable en el ecosistema .NET (Microsoft, 2025a) requiere decisiones arquitectónicas desde las etapas tempranas del diseño. La arquitectura típica de un sistema auditable en .NET incluye al menos los siguientes componentes:

- ✓ Logger configurado centralmente: Se refiere a una instancia de la biblioteca Serilog (Serilog Contributors, n.d.) configurada al inicio de la aplicación, típicamente en el método Program.cs de aplicaciones ASP.NET Core. La configuración debe especificar múltiples "sinks" que son destinos de registro de Log, que pueden ser: archivo local, base de datos centralizada para auditoría, servicio SIEM para monitoreo de seguridad.
- ✓ Middleware de auditoría: En aplicaciones web ASP.NET Core, un middleware personalizado puede capturar automáticamente información de cada request HTTP, que puede incluir información como: usuario autenticado, dirección IP, endpoint invocado, parámetros de entrada, timestamp de inicio y fin, código de respuesta HTTP.
- ✓ Interceptores a nivel de datos: La librería Entity Framework Core, es el ORM estándar de facto en .NET, proporciona "hooks" para interceptar operaciones de base de datos. Para el efecto se implementa SaveChangesInterceptor, que permite capturar automáticamente todas las operaciones de INSERT, UPDATE, DELETE, registrando el estado anterior y posterior de las entidades afectadas en la base de datos.
- ✓ Servicios de auditoría especializados: Usados para operaciones de negocio críticas que requieren registros de auditoría detallados, es recomendable implementar estos servicios dedicados que encapsulen la lógica de auditoría y aseguren consistencia en el registro de eventos.

Un principio arquitectónico fundamental es que las operaciones de auditoría deben ejecutarse de manera asíncrona siempre que sea posible, para minimizar el impacto en el rendimiento de las

operaciones de negocio principales.

En este capítulo únicamente se aborda la implementación del componente Logger usando la librería especializada Serilog.

7.2 Guía de Implementación de Serilog en .NET Core

Serilog para .NET Core es una librería que se especializa en generación de Registros de Auditoría a través de logging (Serilog Contributors, n.d.), para ello implementa "buffering" y "batching" automáticos para la mayoría de sus sinks, reduciendo significativamente la latencia. Buffering se refiere a mantener temporalmente los eventos de log en memoria antes de enviarlos a su destino final, con el objetivo de desacoplar la generación de logs del proceso de escritura y evitar impactos de rendimiento cuando el destino es lento o está momentáneamente indisponible. Batching, en cambio, consiste en agrupar varios eventos ya almacenados en el buffer y enviarlos juntos en una sola operación, reduciendo llamadas de I/O, uso de red o costos de escritura. En la práctica, Serilog primero bufferiza los eventos y luego los despacha en lotes según tamaño o intervalo de tiempo configurado. Además, es de configuración sencilla y eficiente trabajo en todas las capas de un sistema auditable: Captura, Enriquecimiento, Almacenamiento, Protección y Análisis.

Para su implementación, primeramente, se debe definir el contenido estructural del logging priorizando Eventos de Negocio vs. Eventos Técnicos, por ejemplo, distinguir entre eventos que tienen significado de negocio ("Usuario aprobó orden de compra por $50,000") y eventos puramente técnicos ("Conexión a base de datos establecida"). Los primeros son esenciales para auditoría; los segundos son útiles para debugging, pero deben registrarse a niveles de log diferentes para evitar contaminación de los registros de auditoría.

Además de registrar el evento en sí, es importante aportar contexto mediante el enriquecimiento de datos a través de "enrichers". En Serilog, estos permiten agregar de forma automática información contextual a todos los eventos de log, como el nombre de la máquina, el identificador del proceso o del hilo, y datos del entorno de

ejecución. En aplicaciones web, también es habitual usar enrichers personalizados para incluir información relevante del contexto de la solicitud, como el identificador de correlación del request, el usuario autenticado, el tenant en escenarios multi tenant o datos de sesión.

Por otro lado, para que los logs sean realmente útiles al analizarlos, es relevante usar propiedades estructuradas en lugar de concatenar valores dentro de cadenas de texto. Registrar los datos como propiedades, por ejemplo, usando plantillas como {OrderId}, permite realizar búsquedas y consultas eficientes en las plataformas de análisis de logs, algo que no es posible cuando toda la información queda mezclada en un texto plano..

7.2.1 Fundamentos de Configuración mediante appsettings.json

Serilog, originalmente fue diseñado para configurarse mediante API fluida en código C# como parte del Program.cs de un proyecto .NET Core, pero también puede configurarse completamente mediante archivos JSON gracias al paquete nuget Serilog.Settings.Configuration. Este paquete implementa un proveedor de configuración que lee desde Microsoft.Extensions.Configuration, permitiendo que toda la configuración de logging resida en archivos appsettings.json sin necesidad de modificar código compilado.

La ventaja principal de la configuración declarativa radica en la capacidad de modificar el comportamiento de logging en ambientes de producción sin recompilación ni redespliegue de la aplicación. Los cambios a niveles de logging, destinos de escritura, o formatos de salida pueden aplicarse mediante edición de archivo de configuración y, opcionalmente, con recarga en caliente si se habilita reloadOnChange: true en configuración del proveedor JSON.

7.2.2 Instalación de Paquetes Necesarios

La implementación básica requiere el paquete NuGet Serilog.AspNetCore cuya instalación está disponible mediante comando CLI de .NET:

```
1    dotnet add package Serilog.AspNetCore
```

O desde el Package Manager Console de Visual Studio:

```
Install-Package Serilog.AspNetCore
```

El paquete Serilog.AspNetCore incluye automáticamente como dependencia al paquete Serilog.Settings.Configuration, que proporciona el método de extensión ReadFrom.Configuration(). Para funcionalidad adicional de escritura o enriquecimiento, se instalan paquetes específicos de sinks o enrichers según la necesidad. Por ejemplo, para escribir a archivos .log se requiere Serilog.Sinks.File, para consola con colores se usa Serilog.Sinks.Console, y para formato JSON compacto es necesario Serilog.Formatting.Compact.

7.2.3 Integración en Program.cs

La configuración y el uso de Serilog en aplicaciones ASP.NET Core se realizan principalmente a través de llamadas a la clase estática Serilog.Log, la cual actúa como punto central para inicializar el logger, definir los sinks, establecer niveles de logging y aplicar opciones de enriquecimiento. El código mínimo necesario en Program.cs es:

```
//=========================================
// Configurar Serilog leyendo desde
   appsettings.json
//=========================================
var config = new ConfigurationBuilder()
.AddJsonFile("appsettings.json",
optional: false,
reloadOnChange: true)
.Build();

Serilog.Log.Logger = new LoggerConfiguration()
.ReadFrom.Configuration(config)
.CreateLogger();
```

Este código delega completamente la configuración de Serilog al sistema de configuración de ASP.NET Core, que lee automáticamente de appsettings.json y appsettings.{Environment}.json. El método ReadFrom.Configuration() interpreta la sección Serilog del archivo JSON y traduce sus valores a llamadas equivalentes de API fluida.

7.2.4 Estructura de Configuración en appsettings.json

La configuración reside en la sección raíz llamada Serilog del archivo de configuración appsettings.json. La estructura básica contiene cinco subsecciones principales: “MinimumLevel” para control de verbosidad, “WriteTo” para definir destinos de escritura, “Enrich” para añadir propiedades contextuales, “Using” para especificar ensamblados de sinks y enrichers, y “Properties” para adjuntar propiedades globales a todos los eventos.

Un ejemplo de configuración completa pero concisa sería:

```
{
    "Serilog": {
        "Using": [ "Serilog.Sinks.Console", "
           Serilog.Sinks.File" ],
        "MinimumLevel": {
            "Default": "Information",
            "Override": {
                "Microsoft": "Warning",
                "System": "Warning"
            }
        },
        "WriteTo": [
        { "Name": "Console" },
        {
            "Name": "File",
            "Args": { "path": "Logs/log.txt" }
        }
        ],
        "Enrich": [ "FromLogContext", "
           WithMachineName" ],
        "Properties": {
            "Application": "MiAplicacion"
        }
    }
}
```

Configuración de Niveles de Logging: La subsección MinimumLevel controla qué eventos se registran según su severidad. Serilog define seis

niveles en orden creciente de severidad: Verbose, Debug, Information, Warning, Error, y Fatal. El nivel Default establece el mínimo global, mientras que Override permite especificar niveles diferentes para namespaces específicos.

La configuración de override es particularmente útil para suprimir ruido de frameworks como ASP.NET Core o Entity Framework, que generan volúmenes significativos de eventos informativos de bajo valor para auditoría de aplicación. Por ejemplo, establecer "Microsoft": "Warning" elimina eventos informativos de todo el namespace Microsoft.*, conservando solo advertencias, errores y fatales. Los overrides se aplican mediante matching de prefijo, por lo que "Microsoft.EntityFrameworkCore": "Error" es más específico que "Microsoft": "Warning" y tiene precedencia.

Configuración de Sinks mediante WriteTo: Los sinks definen destinos donde los eventos de log serán escritos. La sección WriteTo acepta un array de objetos, cada uno especificando un sink mediante la propiedad Name. Si el sink requiere parámetros, estos se especifican en objeto Args.

Para sink de archivo con rotación diaria y límite de tamaño, la configuración sería:

```
    {
        "Name": "File",
        "Args": {
            "path": "Logs/log-.log",
            "rollingInterval": "Day",
            "fileSizeLimitBytes": 10485760,
            "rollOnFileSizeLimit": true,
            "retainedFileCountLimit": 31,
            "outputTemplate": "{Timestamp:yyyy-MM-
               dd HH:mm:ss.fff zzz} [{Level:u3}] {
               Message:lj}{NewLine}{Exception}"
        }
    }
```

El parámetro rollingInterval: "Day" causa que el archivo se rote diariamente, generando nombres como log-20260111.log. El parámetro

fileSizeLimitBytes establece un tamaño máximo de 10MB por archivo, y rollOnFileSizeLimit: true causa la creación de un nuevo archivo cuando límite es alcanzado. El parámetro retainedFileCountLimit: 31 mantiene solo los últimos 31 archivos, eliminando automáticamente archivos más antiguos.

La plantilla de salida (outputTemplate) controla el formato de cada línea de log. Los tokens entre llaves son propiedades del evento de log: {Timestamp} es marca temporal con formato especificado después de dos puntos, {Level:u3} es nivel en mayúsculas con tres caracteres de ancho, {Message:lj} es mensaje con formato JSON para valores estructurados, y {Exception} es excepción si estuviere presente.

Enriquecimiento de Eventos: Los enrichers añaden propiedades adicionales a cada evento de log. Algunos enrichers comunes incluyen FromLogContext que habilita contexto de logging dinámico, WithMachineName que añade nombre de máquina, WithThreadId que añade identificador de thread, y WithEnvironmentName que añade nombre de ambiente de ASP.NET Core.

Para utilizar enrichers, primero se instalan otros paquetes correspondientes mediante el CLI de .NET:

```
dotnet add package Serilog.Enrichers.
   Environment
dotnet add package Serilog.Enrichers.Thread
```

O desde el Package Manager Console:

```
Install-Package Serilog.Enrichers.Environment
Install-Package Serilog.Enrichers.Thread
```

Luego se especifican en configuración mediante array de nombres que corresponden a métodos de extensión (sin prefijo Enrich.):

```
"Enrich": [
"FromLogContext",
"WithMachineName",
"WithThreadId",
```

```
    "WithEnvironmentName"]
```

Las propiedades añadidas por enrichers están disponibles en plantillas de salida y son incluidas automáticamente en formatos estructurados como JSON. Por ejemplo, con estos enrichers activos, cada evento de log tendrá propiedades MachineName, ThreadId, y EnvironmentName que pueden referenciarse en outputTemplate como {MachineName}, {ThreadId}, y {EnvironmentName}.

Propiedades Globales: La sección Properties permite adjuntar propiedades constantes a todos los eventos. Esto es útil para identificar aplicación, versión, o ambiente cuando logs de múltiples fuentes se agregan en sistema centralizado. Por ejemplo:

```
    "Properties": {
        "Application": "SistemaAuditoria",
        "Version": "2.1.0",
        "Environment": "Production"
    }
```

7.2.5 Configuración Específica por Ambiente

ASP.NET Core soporta múltiples archivos de configuración cargados en cascada según ambiente de ejecución. El archivo appsettings.json contiene configuración base, mientras appsettings.Development.json, appsettings.Staging.json, y appsettings.Production.json contienen overrides específicos de ambiente.

Para logging, esto permite configurar verbosidad mayor en desarrollo que en producción. Por ejemplo, appsettings.Development.json podría tener:

```
    {
        "Serilog": {
            "MinimumLevel": {
                "Default": "Debug",
                "Override": {
                    "Microsoft": "Information"
                }
```

```
            },
            "WriteTo": [
            { "Name": "Console" }
            ]
        }
    }
```

Mientras appsettings.Production.json podría ser:

```
    {
        "Serilog": {
            "MinimumLevel": {
                "Default": "Warning"
            },
            "WriteTo": [
            {
                "Name": "File",
                "Args": {
                    "path": "/var/log/aplicacion/
                       log-.txt",
                    "rollingInterval": "Day"
                }
            }
            ]
        }
    }
```

7.2.6 Formato Estructurado mediante JSON Compacto

Para integración con sistemas de agregación de logs como Elasticsearch o Seq, es preferible escribir eventos en formato JSON estructurado en lugar de texto plano. En este caso, el paquete Serilog.Formatting.Compact proporciona un formatter que serializa eventos a JSON compacto, una línea por evento.

Instalar el paquete desde CLI de .NET:

```
    dotnet add package Serilog.Formatting.Compact
```

O desde el Package Manager Console:

```
Install-Package Serilog.Formatting.Compact
```

La configuración especifica el formateador de JSON compacto mediante la propiedad formatter de la sección Serilog / WriteTo / Args:

```
{
    "Name": "File",
    "Args": {
        "path": "Logs/log-.json",
        "rollingInterval": "Day",
        "formatter": "Serilog.Formatting.
           Compact.CompactJsonFormatter,
           Serilog.Formatting.Compact"
    }
}
```

Cada línea Log del archivo resultante es un objeto JSON válido con la siguiente estructura:

```
{"@t":"2026-01-11T15:30:45.1234567Z","@l":"
   Information","@mt":"Usuario {UserId} accedi
   ó a recurso {ResourceId}","UserId":123,"
   ResourceId":"DOC-456"}
```

Las propiedades con prefijo @ son metadata de Serilog: @t es timestamp ISO 8601, @l es nivel, @mt es plantilla de mensaje. Las propiedades sin prefijo son valores estructurados del evento.

7.2.7 Recarga Dinámica de Configuración

Para habilitar la recarga de configuración sin reiniciar la aplicación, se especifica el parámetro reloadOnChange:true cuando se carga el archivo JSON, nótese el ejemplo mínimo de código en Program.cs:

```
//=====================================================

// Configurar Serilog leyendo desde
   appsettings.json
```

```
    //=====================================================

    var config = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json",
    optional: false,
    reloadOnChange: true)
    .Build();

    Serilog.Log.Logger = new LoggerConfiguration()
    .ReadFrom.Configuration(config)
    .CreateLogger();
```

Con esta configuración, los cambios a valores en secciones MinimumLevel.Default, MinimumLevel.Override, y switches de nivel son detectados y aplicados automáticamente. Sin embargo, los cambios a sinks (sección WriteTo) no se aplican dinámicamente, requieren reinicio de la aplicación.

7.2.8 Uso de Serilog según Nivel de Verbosidad

Serilog soporta seis niveles de logging en orden creciente de severidad: Verbose, Debug, Information, Warning, Error, y Fatal. La elección del nivel apropiado es crítica para que el filtrado mediante MinimumLevel funcione correctamente y para mantener utilidad de los logs sin generar volumen excesivo.

Existen dos formas de escribir eventos de log: mediante la clase estática Serilog.Log, o mediante la interfaz ILogger<T>de Microsoft.Extensions.Logging inyectada por dependencia. Ambas interfaces utilizan la misma infraestructura subyacente de Serilog configurada, pero difieren en sus métodos de logging.

API del logger estático de Serilog:

```
    Using Serilog;

    Log.Verbose("mensaje", params);
    Log.Debug("mensaje", params);
    Log.Information("mensaje", params);
    Log.Warning("mensaje", params);
    Log.Error(exception, "mensaje", params);
```

```
8       Log.Fatal(exception, "mensaje", params);
```

API de ILogger<T>de Microsoft.Extensions.Logging:

```
1       _logger.LogTrace("mensaje", params);
2       _logger.LogDebug("mensaje", params);
3       _logger.LogInformation("mensaje", params);
4       _logger.LogWarning("mensaje", params);
5       _logger.LogError(exception, "mensaje", params)
            ;
6       _logger.LogCritical(exception, "mensaje",
            params);
```

La documentación oficial recomienda utilizar ILogger<T>en lugar de la API estática de Serilog cuando se trabaja en ASP.NET Core, porque hace el código más estándar y portable entre diferentes proveedores de logging. Sin embargo, ambas APIs son funcionalmente equivalentes cuando Serilog está configurado como proveedor de logging. Con este antecedente, a continuación, se muestran ejemplos de uso para ambos escenarios, según nivel de verbosidad.

Verbose: Diagnóstico Exhaustivo

El nivel Verbose es el más detallado de todos y está pensado para situaciones de análisis profundo, como tareas de tracing o depuración a nivel interno de la aplicación. Se utiliza cuando es necesario entender con precisión cómo se está ejecutando el código, ya que registra cada paso relevante del flujo de ejecución y los valores intermedios de las variables. Tal como indica la documentación oficial, este nivel ofrece una visibilidad completa del comportamiento interno del sistema, pero debe usarse con cautela debido al gran volumen de información que genera.

Ejemplo con logger estático:

```
1       Log.Verbose("Validación para UserId: {UserId}"
            , userId);
2       Log.Verbose("Regla {rId} evaluada: {Result}",
            rId, result);
```

Con ILogger<T>:

```
_logger.LogTrace("Validación para UserId: {UserId}", userId);
_logger.LogTrace("Regla {rId} evaluada: {Result}", rId, result);
```

Debug: Información de Desarrollo

El nivel Debug se utiliza para registrar información orientada al desarrollo y al diagnóstico de problemas. Captura eventos internos del sistema que normalmente no son visibles desde el exterior, pero que resultan útiles para entender cómo se ejecutó una operación o por qué se produjo determinado comportamiento. Este nivel permite analizar el flujo interno de la aplicación sin llegar al nivel de detalle extremo del modo Verbose.

Ejemplo con logger estático:

```
Log.Debug("Usuario {UserId} autenticado. Roles: {Roles}", userId, string.Join(",", roles));

Log.Debug("Query ejecutado en {ElapsedMs}ms", elapsed.TotalMilliseconds);
```

Con ILogger<T>:

```
_logger.LogDebug("Usuario {UserId} autenticado. Roles: {Roles}", userId, string.Join(",", roles));

_logger.LogDebug("Query ejecutado en {ElapsedMs}ms", elapsed.TotalMilliseconds);
```

Information: Eventos de Negocio Normales

El nivel Information se utiliza para registrar los eventos habituales del sistema, aquellos que reflejan el funcionamiento normal de la aplicación y el cumplimiento de sus responsabilidades de negocio. Incluye acciones esperadas como el inicio o finalización de procesos,

operaciones completadas con éxito o flujos de trabajo estándar. Este suele ser el nivel mínimo predeterminado cuando no se define una configuración explícita, ya que ofrece un buen equilibrio entre visibilidad y volumen de información.

Ejemplo con logger estático:

```
Log.Information("Orden {OrderId} creada por
   usuario {UserId}. Monto: {Amount:C}",
   orderId, userId, amount);

Log.Information("Pago procesado. TransactionId
   : {TransactionId}, Status: {Status}",
   transactionId, status);
```

Con ILogger<T>:

```
_logger.LogInformation("Orden {OrderId} creada
    por usuario {UserId}. Monto: {Amount:C}",
   orderId, userId, amount);

_logger.LogInformation("Pago procesado.
   TransactionId: {TransactionId}, Status: {
   Status}", transactionId, status);
```

Warning: Degradación o Situaciones Anómalas

El nivel Warning se utiliza para señalar condiciones que no son críticas, pero que indican que el sistema no está funcionando de forma ideal. Refleja situaciones anómalas, comportamientos fuera de los parámetros esperados o estados de degradación que, aunque no detienen el servicio, requieren atención para evitar que evolucionen en problemas más graves.

Ejemplo con logger estático:

```
Log.Warning("Intento de acceso a recurso {
   ResourceId} denegado para usuario {UserId}"
   , resourceId, userId);

Log.Warning("API externa {ApiName} respondió {
```

```
        StatusCode}. Reintentando", apiName,
        statusCode);
```

Con ILogger<T>:

```
1       _logger.LogWarning("Intento de acceso a
            recurso {ResourceId} denegado para usuario
            {UserId}", resourceId, userId);
2
3       _logger.LogWarning("API externa {ApiName}
            respondió {StatusCode}. Reintentando",
            apiName, statusCode);
```

Error: Fallos de Funcionalidad

El nivel Error se emplea cuando una operación o funcionalidad no se ejecuta como se esperaba y el resultado no puede considerarse correcto. Aunque la aplicación puede seguir funcionando, el evento registrado indica claramente que algo falló en ese contexto específico, proporcionando información clave para el análisis y la corrección del problema.

Ejemplo con logger estático:

```
1       try
2       {
3           await ProcessPayment(orderId);
4       }
5       catch (PaymentGatewayException ex)
6       {
7           Log.Error(ex, "Falló procesamiento de pago
                para orden {OrderId}. Gateway: {
               Gateway}",
8           orderId, ex.Gateway);
9       }
```

Con ILogger<T>:

```
1       try
2       {
3           await ProcessPayment(orderId);
```

```
    }
    catch (PaymentGatewayException ex)
    {
        _logger.LogError(ex, "Falló procesamiento
            de pago para orden {OrderId}. Gateway:
            {Gateway}",
        orderId, ex.Gateway);
    }
```

Fatal: Errores Catastróficos

El nivel Fatal representa las fallas más graves que puede experimentar un sistema. Se utiliza cuando ocurre un error crítico que provoca, o está a punto de provocar, la terminación de la aplicación. En el contexto de ILogger<T>, este nivel se denomina Critical y suele indicar condiciones que requieren intervención inmediata, ya que comprometen por completo la continuidad del servicio.

Ejemplo con logger estático:

```
    try
    {
        InitializeCriticalDependency();
    }
    catch (Exception ex)
    {
        Log.Fatal(ex, "Falló inicialización de {
            DependencyName}. Aplicación no puede
            continuar",
        dependencyName);
        throw;
    }
```

Con ILogger<T>:

```
    try
    {
        InitializeCriticalDependency();
    }
    catch (Exception ex)
    {
```

```
7           _logger.LogCritical(ex, "Falló
                inicialización de {DependencyName}.
                Aplicación no puede continuar",
8           dependencyName);
9           throw;
10      }
```

Logging Estructurado con Message Templates

Serilog utiliza message templates para registrar información de forma estructurada y no solo como texto plano. En estos templates, los nombres de las propiedades se escriben entre llaves y los valores reales se pasan como parámetros adicionales al método de logging. De esta manera, el mensaje sigue siendo legible para las personas, pero al mismo tiempo los datos quedan capturados como campos independientes, lo que facilita su posterior búsqueda, filtrado y análisis en sistemas de almacenamiento y monitoreo de logs.

Correcto - Logging estructurado:

```
1       Log.Information("Usuario {UserId} modificó
            configuración. Cambios: {@Changes}",
2       userId, changesObject);
```

Incorrecto - Concatenación de strings:

```
1       Log.Information("Usuario " + userId + "
            modificó configuración");
```

El prefijo @ en {@Changes} indica destructuración del objeto, capturando sus propiedades en lugar de solo invocar ToString(). Esto permite queries sobre propiedades individuales en sistemas de agregación de logs.

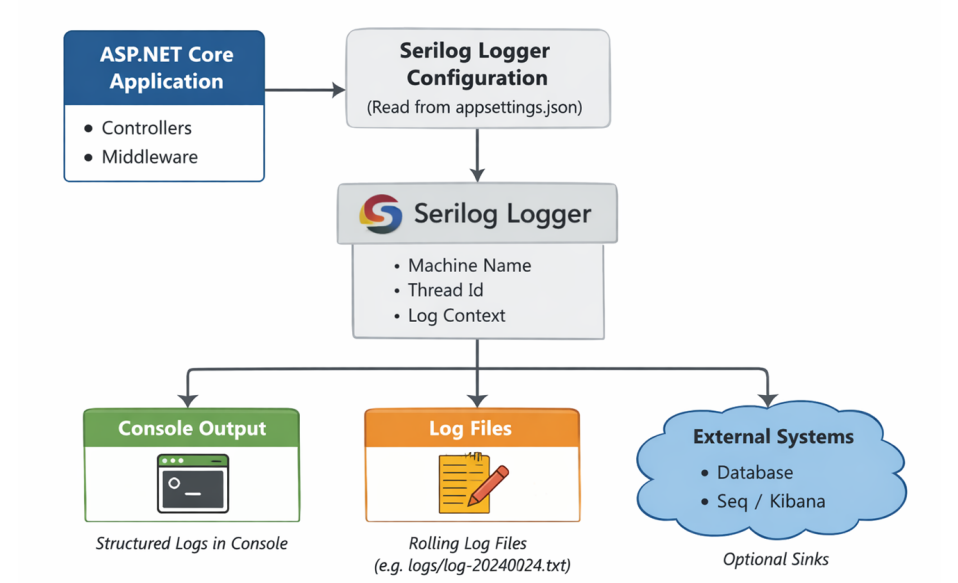
7.2.9 Ejemplo de Uso de Serilog en un proyecto .NET Core

Se inicia definiendo la arquitectura del proyecto, para ello sirve de apoyo la Figura 5, donde se representa la integración de Serilog dentro de una aplicación ASP.NET Core, mostrando el flujo completo de generación, procesamiento y almacenamiento de los logs.

En el extremo izquierdo se encuentra la aplicación ASP.NET Core, específicamente sus componentes principales: controladores y middleware. Estos son los puntos donde se generan los eventos de logging, ya sea de forma explícita (mediante llamadas a ILogger) o implícita (como en el caso del middleware de registro de solicitudes HTTP).

Estos eventos son dirigidos hacia la configuración central de Serilog, la cual se carga desde el archivo appsettings.json. Esta configuración define el comportamiento del sistema de logging, incluyendo niveles de severidad, destinos de salida y reglas específicas para distintos componentes. Este enfoque permite modificar el comportamiento sin necesidad de recompilar la aplicación.

Figura5:
Arquitectura de una aplicación .Net Core usando Serilog



Nota: Elaboración Propia

A partir de esta configuración, se instancia el núcleo del sistema: el logger de Serilog. En esta etapa se aplican los denominados enrichers, que añaden información contextual a cada evento de log, como

el nombre de la máquina, el identificador del hilo de ejecución o datos adicionales del contexto de la aplicación. Este enriquecimiento transforma los logs en información estructurada y altamente útil para análisis posteriores.

Finalmente, los eventos procesados son distribuidos hacia múltiples destinos de salida, conocidos como sinks. En la figura se destacan tres principales:

- ✓ La consola, utilizada principalmente en entornos de desarrollo o monitoreo en tiempo real.
- ✓ Archivos de log, con rotación automática, que permiten persistencia local y trazabilidad histórica.
- ✓ Sistemas externos, como bases de datos o plataformas de observabilidad (por ejemplo, Seq o Kibana), que facilitan análisis avanzado, visualización y centralización de logs.

Este flujo evidencia una arquitectura desacoplada, donde la generación de eventos, su procesamiento y su almacenamiento están claramente separados, permitiendo escalabilidad, mantenibilidad y flexibilidad en la gestión del registro de eventos dentro de la aplicación1,

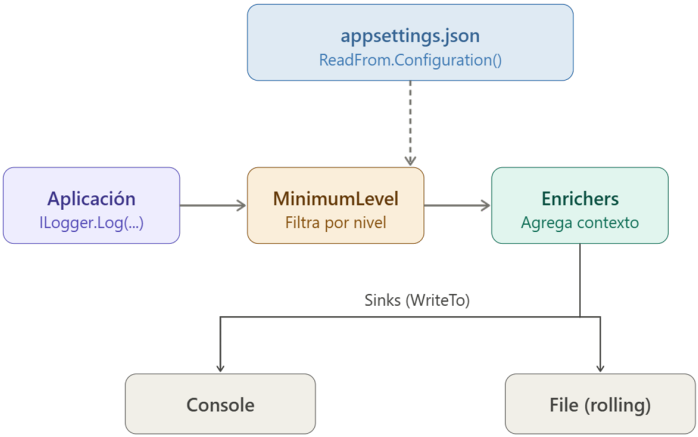
Adicionalmente, el flujo de procesamiento de eventos en Serilog, tal como se ilustra en la Figura 6, responde a un modelo secuencial y declarativo en el que cada etapa cumple una función específica dentro del ciclo de vida del log.

El punto de entrada del pipeline está constituido por la aplicación, donde se generan los eventos mediante invocaciones a mecanismos de logging, típicamente a través de la abstracción ILogger. Cada invocación produce un evento que contiene un mensaje, un nivel de severidad y, opcionalmente, datos estructurados asociados.

Una vez generado, el evento es dirigido hacia el sistema de configuración, el cual ha sido previamente definido en el archivo appsettings.json y cargado mediante el método ReadFrom.Configuration(). Esta etapa no transforma el evento en sí, sino que establece las reglas

que gobernarán su procesamiento a lo largo del pipeline.

Figura 6:
Pipeline de flujo de Serilog



Nota: Elaboración Propia

El primer filtro aplicado corresponde al bloque MinimumLevel. En esta fase, el sistema evalúa la severidad del evento y determina si debe continuar su procesamiento o ser descartado. Este mecanismo permite controlar la cantidad de información registrada, evitando la sobrecarga de logs con eventos irrelevantes y priorizando aquellos de mayor importancia operativa.

Superado el filtro de nivel, el evento ingresa a la etapa de enriquecimiento (Enrichers). Aquí se incorporan datos contextuales adicionales que no estaban presentes en el momento de la generación del evento. Estos pueden incluir información del entorno de ejecución, como el nombre de la máquina, el identificador del hilo o propiedades del contexto de la solicitud. El resultado es un evento más completo, con mayor valor semántico para análisis posteriores.

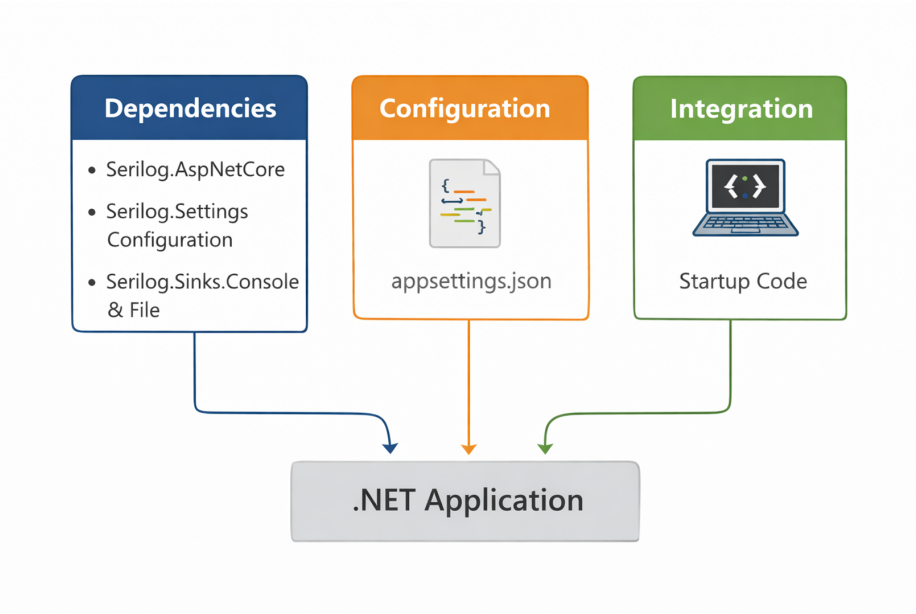
Finalmente, el evento enriquecido es distribuido hacia uno o varios destinos de salida, definidos en la sección WriteTo del archivo de configuración. Estos destinos, conocidos como sinks, representan los

puntos finales del pipeline. En el escenario mostrado, se incluyen la consola y archivos con rotación automática. Cada sink recibe el evento de forma independiente, permitiendo múltiples formas de persistencia o visualización sin interferencia entre ellas.

Este pipeline evidencia un diseño basado en responsabilidades claramente separadas: generación, filtrado, enriquecimiento y persistencia. La configuración externa mediante appsettings.json permite modificar el comportamiento de cada etapa sin alterar el código fuente, consolidando un enfoque flexible y alineado con las mejores prácticas de observabilidad en aplicaciones modernas.

La implementación se fundamenta en tres elementos esenciales: la inclusión de dependencias, la configuración declarativa mediante archivo externo y la integración en el ciclo de vida de la aplicación. Ver Figura 7.

Figura 7:
Componentes fundamentales para la implementación de Serilog en aplicaciones .NET



Nota: Elaboración Propia

En primer lugar, se incorporan las dependencias necesarias en el archivo de proyecto. Estas incluyen el paquete principal de integración con ASP.NET Core, así como extensiones que permiten leer configuraciones desde archivos JSON y definir destinos de salida, tales como la consola y archivos persistentes. Este enfoque modular permite extender fácilmente el sistema de logging sin alterar la lógica de negocio.

Los siguientes son los comandos para incluir las dependencias:

```
dotnet add package Serilog.AspNetCore --
   version 8.0.0
dotnet add package Serilog.Settings.
   Configuration --version 8.0.0
dotnet add package Serilog.Sinks.Console --
   version 5.0.1
dotnet add package Serilog.Sinks.File --
   version 5.0.0
```

A continuación, se establece la configuración en el archivo appsettings.json. Este mecanismo desacopla la definición del comportamiento del sistema de registro del código fuente, favoreciendo la mantenibilidad y la adaptabilidad a distintos entornos. En esta configuración se definen aspectos clave como el nivel mínimo de registro, que determina la severidad de los eventos a capturar, así como reglas específicas para reducir la verbosidad de componentes internos del framework. Asimismo, se especifican los destinos de salida mediante la sección WriteTo, donde se configura tanto la emisión de eventos a la consola como su persistencia en archivos rotativos diarios. El uso de plantillas de salida permite estandarizar el formato de los registros, facilitando su posterior análisis.

El siguiente es el contenido mínimo del archivo appsetting.json:

```
{
    "Serilog": {
        "Using": [ "Serilog.Sinks.Console", "
           Serilog.Sinks.File" ],
        "MinimumLevel": {
            "Default": "Information",
```

```
            "Override": {
                "Microsoft": "Warning",
                "System": "Warning"
            }
        },
        "WriteTo": [
        {
            "Name": "Console"
        },
        {
            "Name": "File",
            "Args": {
                "path": "logs/log-.txt",
                "rollingInterval": "Day",
                "outputTemplate": "[{Timestamp
                    :yyyy-MM-dd HH:mm:ss} {
                    Level:u3}] {Message:lj}{
                    NewLine}{Exception}"
            }
        }
        ],
        "Enrich": [ "FromLogContext", "
            WithMachineName", "WithThreadId" ]
    }
}
```

Posteriormente, la integración se realiza en el punto de entrada de la aplicación, específicamente en el archivo Program.cs. Allí se construye una instancia del logger a partir de la configuración previamente definida, asegurando que todas las características establecidas en el archivo JSON sean respetadas. Esta instancia se registra en el host de la aplicación, reemplazando el sistema de logging por defecto. De este modo, cualquier componente que dependa de la abstracción de logging proporcionada por el framework utilizará automáticamente Serilog como proveedor subyacente.

Adicionalmente, se incorpora un middleware especializado que permite registrar automáticamente las solicitudes HTTP entrantes. Esta funcionalidad resulta especialmente útil para el monitoreo del

tráfico y la identificación de patrones de uso o posibles fallos en la interacción cliente-servidor.

El siguiente es el contenido del archivo Program.cs:

```
using Serilog;

var builder = WebApplication.CreateBuilder(
   args);

// Configurar Serilog desde appsettings.json
Log.Logger = new LoggerConfiguration()
.ReadFrom.Configuration(builder.Configuration)
.Enrich.FromLogContext()
.CreateLogger();

builder.Host.UseSerilog();
builder.Services.AddControllers();
var app = builder.Build();

// Middleware de logging automático de
   requests
app.UseSerilogRequestLogging();
app.MapControllers();
app.Run();
```

Finalmente, en los controladores de la aplicación, se emplea la interfaz de logging inyectada por dependencia para registrar eventos de interés. En el ejemplo presentado, se ilustra tanto el registro de información general como la captura de errores mediante bloques de manejo de excepciones. Este enfoque garantiza que los eventos relevantes queden documentados con suficiente contexto, lo cual es esencial para tareas de depuración y auditoría.

El siguiente es el contenido del archivo del controlador TestController.cs:

```
using Microsoft.AspNetCore.Mvc;

namespace SerilogDemo.Controllers;

```

```
    [ApiController]
    [Route("[controller]")]
    public class TestController : ControllerBase
    {
        private readonly ILogger<TestController>
            _logger;

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

        [HttpGet]
        public IActionResult Get()
        {
            _logger.LogInformation("Endpoint GET
                ejecutado Ok");
            try
            {
                int x = 0;
                int y = 10 / x; // Forzar error
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ocurrió un
                    error de prueba");
            }
            return Ok("Revisa consola y archivo de
                logs");
        }
    }
```

Finalmente, se debe entender que la implementación de un sistema de registro estructurado mediante Serilog en aplicaciones ASP.NET Core representa un paso decisivo hacia la construcción de soluciones observables, mantenibles y preparadas para entornos productivos. A lo largo de este capítulo se ha evidenciado cómo, a través de una adecuada incorporación de dependencias, una configuración

declarativa centralizada y una integración coherente en el ciclo de vida de la aplicación, es posible establecer un mecanismo de logging robusto sin introducir complejidad innecesaria en el código.

El enfoque adoptado permite no solo capturar eventos, sino enriquecerlos y dirigirlos de manera flexible hacia múltiples destinos, facilitando su análisis y explotación. Esta capacidad de adaptación, sustentada en la configuración externa, garantiza que el sistema pueda evolucionar conforme cambian los requerimientos operativos, sin afectar la estabilidad de la aplicación.

De esta manera, el registro de eventos deja de ser un elemento accesorio para convertirse en un componente estratégico dentro de la arquitectura del software, habilitando prácticas como la trazabilidad, la auditoría y el monitoreo continuo. La correcta implementación de estas capacidades sienta las bases para niveles superiores de madurez, donde la observabilidad se integra de forma natural en el diseño y operación de los sistemas.

Capítulo 8

CONCLUSIONES

A lo largo de este libro, se ha explorado el concepto de software auditable desde múltiples perspectivas, teórica, técnica, regulatoria y práctica. Se ha visto cómo la auditabilidad no es meramente una característica técnica que puede ser añadida superficialmente a sistemas existentes, sino una propiedad fundamental que debe ser considerada desde las primeras etapas de diseño arquitectónico y mantenida disciplinadamente a través de todo el ciclo de vida del software.

Los fundamentos teóricos establecen que el software verdaderamente auditable requiere arquitectura en capas que separa responsabilidades de captura de eventos, enriquecimiento de contexto, almacenamiento inmutable, protección criptográfica, y análisis forense. El principio de "separation of duty" aplicado a auditoría dicta que los sistemas siendo auditados no deben tener control sobre sus propios registros de auditoría, requiriendo componentes independientes que actúan como registradores confiables.

La implementación efectiva de auditabilidad requiere un enfoque sistemático que abarque múltiples dimensiones: arquitectura de sistemas que facilite la captura de eventos relevantes mediante patrones como interceptores y event sourcing, almacenamiento seguro e inmutable de registros utilizando bases de datos append-only y estructuras de datos protegidas criptográficamente, mecanismos de protección que garanticen integridad y no repudio mediante firmas digitales y timestamping, capacidades de análisis y consulta que permitan investigaciones forenses y detección de anomalías, y políticas de retención y archivado que satisfagan requisitos regulatorios.

Los marcos regulatorios: GDPR, HIPAA, SOX, PCI-DSS, establecen requisitos legalmente vinculantes para auditoría y demostración de cumplimiento. Organizaciones que operan en múltiples jurisdicciones o sectores deben cumplir obligaciones superpuestas y ocasionalmente conflictivas. Sin embargo, estos marcos también proporcionan una guía valiosa sobre lo que constituye auditoría efectiva.

Los casos de uso verticales ilustran cómo requisitos de auditoría se manifiestan en contextos específicos. El sector financiero requiere

orden de registros de auditoría con precisión de microsegundos. Healthcare requiere capacidad de rastrear cada acceso a información sensible de pacientes. La facturación electrónica en Ecuador requiere firma digital con XAdES-BES y trazabilidad completa del ciclo de vida de comprobantes. Cada dominio tiene matices únicos que deben ser comprendidos y abordados.

Mirando hacia el futuro, varios involucrados están moldeando la evolución de software auditable. La proliferación de sistemas de inteligencia artificial presenta desafíos únicos para la auditabilidad debido a la opacidad de modelos de machine learning. La adopción creciente de arquitecturas de microservicios y serverless aumenta la complejidad de mantener trazabilidad end-to-end. Las regulaciones emergentes como el AI Act europeo están expandiendo el alcance de requisitos de auditabilidad más allá de dominios tradicionales.

El mensaje fundamental es que en un mundo donde el software persiste crecientemente en toda actividad humana organizada, la capacidad de auditar y verificar el comportamiento de software no es lujo sino necesidad. El software que no puede ser auditado efectivamente no puede ser confiado para operar en contextos críticos. Como desarrolladores, arquitectos y líderes tecnológicos, se tiene la responsabilidad profesional y ética de construir sistemas que sean no solo funcionales y eficientes, sino también transparentes, trazables y auditables.

El software auditable ha evolucionado de ser una consideración secundaria en el desarrollo de sistemas a constituir un requisito fundamental en el diseño de aplicaciones modernas. Esta evolución refleja la creciente madurez de la industria del software y el reconocimiento de que la transparencia y la rendición de cuentas son aspectos tan críticos como la funcionalidad y el rendimiento.

Los marcos regulatorios como GDPR y HIPAA han elevado significativamente el nivel de exigencia, requiriendo no solo la implementación de controles de auditoría sino la capacidad de demostrar cumplimiento de manera proactiva. Esta tendencia regulatoria probablemente se intensificará, particularmente con la emergencia de

regulación específica para sistemas de inteligencia artificial.

En el contexto del ecosistema .NET, herramientas como Serilog para logging estructurado, las capacidades de auditoría nativas de SQL Server y Entity Framework Core, y los frameworks de ASP.NET Core proporcionan una base técnica sólida para implementar software auditable. Sin embargo, la tecnología por sí sola es insuficiente; se requiere una comprensión profunda de los principios subyacentes y un compromiso organizacional con la transparencia.

Bibliografía

Adams, C., Cain, P., Pinkas, D., & Zuccherato, R. (2001). *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)* (inf. téc. N.º RFC 3161). IETF. https://doi.org/10.17487/RFC3161

Apache Software Foundation. (n.d.). *Kafka Log Implementation*. https://kafka.apache.org/39/implementation/log/

Asamblea Nacional del Ecuador. (n.d.). Código Orgánico Monetario y Financiero, Libro I [Ecuador].

Association of Certified Fraud Examiners. (2024). *Occupational Fraud 2024: A Report to the Nations*. https://www.acfe.com/report-to-the-nations/2024/

Audit trail. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Audit_trail

Bank for International Settlements. (2025). *Global FX trading hits 9.6 trillion per day in April 2025 and OTC interest rate derivatives surge to 7.9 trillion: Triennial Survey*. https://www.bis.org/press/p250930.htm

Bondarenko, P. (2026). Enron scandal. https://www.britannica.com/event/Enron-scandal

Carrillo Mañay, V., Mancero Mosquera, H., & Mancero Rivera, D. S. (2019). Análisis de la crisis bancaria privada ecuatoriana (1994-2000) y sus efectos socioeconómicos. *Cofin Habana*. http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2073-60612019000300017

Contraloría General del Estado. (2017). Contraloría General del Estado. https://www.contraloria.gob.ec/CentralMedios/PrensaDia/15946

Contraloría General del Estado. (2020). Contraloría General del Estado. https://www.contraloria.gob.ec/CentralMedios/SalaPrensa/23824

Contraloría General del Estado. (n.d.). Normas de Control Interno [Estado: Vigente]. https://www.lexis.com.ec

DataSunrise. (2025). ¿Para qué se utiliza la auditoría de datos? Aspectos esenciales de seguridad. https://www.datasunrise.com/es/informacion-profesional/para-que-se-utiliza-la-auditoria-de-datos/?utm_source=chatgpt.com

de Derechos y Justicia, O. (s.f.). CORRUPCIÓN EN TIEMPOS DE COVID: LA OTRA PANDEMIA EN ECUADOR. http://sacc.corteconstitucional.gob.ec/storage/api/v1/10_DWL_FL/e2NhcnBldGE6J3RyYW1pdGUnLCB1dWlkOidlM

Dijkstra, E. W. (s.f.). *The Humble Programmer (EWD 340)*. https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html

EFE. (2025). Odebrecht, una larga lista de países y políticos latinoamercanos relacionados con sobornos. https://www.swissinfo.ch/spa/odebrecht,-una-larga-lista-de-pa%C3%ADses-y-pol%C3%ADticos-latinoamercanos-relacionados-con-sobornos/89173496

Elmasri, R., & Navathe, S. B. (2004). *Fundamentals of Database Systems* (4.ª ed.). Addison-Wesley.

Erazo, V. (2022). *23 años del feriado bancario en Ecuador*. Wambra Medio Comunitario. https://wambra.ec/23-anos-feriado-bancario-ecuador/

European Union. (2016a). Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo (GDPR). https://eur-lex.europa.eu/legal-content/ES/ALL/?uri=uriserv:OJ.L_.2016.119.01.0001.01.SPA

European Union. (2016b). Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo (GDPR). https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX:32016R0679

Fantl, J. (n.d.). 120922 JustinFantl Working World GS3663834.

Galero, C. G. (2024). El 'caso Odebrecht', una hidra de la corrupción en América Latina. https://www.publico.es/internacional/caso-odebrecht-hidra-corrupcion-america-latina.html

Gotel, O. C. Z., & Finkelstein, A. C. W. (1994). An Analysis of the Requirements Traceability Problem. *Proceedings of the First International Conference on Requirements Engineering*, 94-101.

Grokipedia. (2026). Transaction log. https://grokipedia.com/page/Transaction_log

Guerrero, C. L. (2019). Investigando Lava Jato: el mayor escándalo de corrupción unió a los periodistas en América Latina. https://gijn.org/es/articulos/investigando-lava-jato-el-mayor-escandalo-de-corrupcion-unio-a-los-periodistas-en-america-latina/

Haber, S., & Stornetta, W. S. (1991). How to Time-Stamp a Digital Document. *Journal of Cryptology*, *3*(2), 99-111. https://doi.org/10.1007/BF00196791

IBM. (n.d.). *Db2 for Linux, UNIX and Windows Documentation*. https://www.ibm.com/docs/en/db2/11.5.x

Institute of Electrical and Electronics Engineers. (1997). IEEE Std 1028-1997 - IEEE Standard for Software Reviews. https://ieeexplore.ieee.org/document/663254

Institute of Electrical and Electronics Engineers. (2008). IEEE Std 1028-2008 - IEEE Standard for Software Reviews and Audits. https://standards.ieee.org/ieee/1028/4402/

International Organization for Standardization. (2011). *ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. https://www.iso.org/standard/35733.html

International Organization for Standardization. (2013). ISO/IEC 27001:2013 Information technology — Security techniques —

Information security management systems — Requirements. https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en

ISACA. (n.d.). *COBIT: Control Objectives for Information and Related Technologies*. https://www.isaca.org/resources/cobit

Kent, K., & Souppaya, M. (2006). *Guide to Computer Security Log Management: Recommendations of the National Institute of Standards and Technology* (inf. téc. N.º NIST Special Publication 800-92). National Institute of Standards y Technology. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf

Krasner, H. (2022). Cost of Poor Software Quality in the U.S.: A 2022 Report. https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/

Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, *21*, 558-565. https://doi.org/10.1145/359545.359563

Levin Center for Oversight and Democracy. (n.d.). Congress and the Enron Scandal. Consultado el 31 de marzo de 2026, desde https://levin-center.org/what-is-oversight/portraits/congress-and-the-enron-scandal/

Ley Orgánica de Empresas Públicas [Ecuador, 16 de octubre de 2009]. (2009).

MacKenzie, D. A. (2001). *Mechanizing Proof: Computing, Risk, and Trust*.

McCarroll, G. D., & Rowley, B. A. (1979). An Investigation of the Existence of Electrically Located Acupuncture Points. *IEEE Transactions on Biomedical Engineering*, *BME-26*(3), 177-181. https://doi.org/10.1109/TBME.1979.326392

McClure, R. M. (2001). NATO Software Engineering Conference 1968.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press.

Merkle, R. C. (1979). *Method of Providing Digital Signatures* [US Patent].

Microsoft. (2025a). *Overview of ASP.NET Core*. https://learn.microsoft.com/en-us/aspnet/core/overview?view=aspnetcore-10.0

Microsoft. (2025b). *Temporal Tables - SQL Server*. https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver17

Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., & Schwarz, P. (1992). ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. *ACM Transactions on Database Systems (TODS)*, *17*, 94-162. https://doi.org/10.1145/128765.128770;WGROUP:STRING:ACM

National Institute of Standards and Technology. (2024). *El Marco de Seguridad Cibernética (CSF) 2.0 del NIST* (inf. téc.). U.S. Department of Commerce. https://doi.org/10.6028/NIST.CSWP.29.spa

O'Neil, P., Cheng, E., Gawlick, D., & O'Neil, E. (1996). The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica*, *33*(4), 351-385. https://doi.org/10.1007/s002360050048

Palmer, G. (2001). A Road Map for Digital Forensic Research [Digital Forensic Research Conference]. *Proceedings of the Digital Forensic Research Workshop (DFRWS)*.

PCI Security Standards Council. (2022). Payment Card Industry Data Security Standard (PCI DSS) [Version 4.0]. https://www.pcisecuritystandards.org/document_library

Perdomo, D. Z., & Mogollón, M. J. (2019). Filanbanco - Hermanos Isaías. https://www.observatorioanticorrupcion.ec/casos-de-corrupcion/filanbanco-hermanos-isaias

Registro Oficial. (2008). Constitución de la República del Ecuador. https://esacc.corteconstitucional.gob.ec/storage/api/v1/10_DWL_FL/e2NhcnBldGE6ICJub3RhaXAyMDIzIiwgdXVpZDoiODJiZWZiNjctZmUxNC00MDRmLTgzMmItYjFjM2RjM2FiODA5LnBkZiJ9

Registro Oficial. (2021). Ley Orgánica de Protección de Datos Personales [Ecuador].

Registro Oficial. (2023). Reglamento a la Ley Orgánica de Protección de Datos Personales [Decreto Ejecutivo No. 904, Ecuador]. https://www.telecomunicaciones.gob.ec/wp-content/uploads/2023/11/Decreto-Ejecutivo-No.-904.pdf

Saltor, C. E. (2013). *La protección de datos personales: estudio comparativo Europa-América con especial análisis de la situación argentina* [Tesis doctoral]. Universidad Complutense de Madrid. https://eprints.ucm.es/22832/1/T34731.pdf

Sarbanes-Oxley Act. (n.d.). Sarbanes-Oxley Compliance Professionals Association (SOXCPA). https://www.sarbanes-oxley-act.com/

Schneier, B., & Kelsey, J. (1998). Cryptographic Support for Secure Logs on Untrusted Machines. *Proceedings of the 7th USENIX Security Symposium*. https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/schneier/schneier.pdf

Schneier, B., & Kelsey, J. (1999). Secure Audit Logs to Support Computer Forensics. *Proceedings of the 1999 USENIX Workshop on Intrusion Detection and Network Monitoring*.

Serilog Contributors. (n.d.). *Serilog Wiki*. https://github.com/serilog/serilog/wiki

Telégrafo, E. (2014). La crisis bancaria de 1999 costó al país 6.170 millones. https://www.eltelegrafo.com.ec/noticias/informacion/1/la-crisis-bancaria-de-1999-costo-al-pais-6-170-millones

Tunggal, A. T., & Ruan, C. (2026). What is SOX Compliance? 2026 Requirements, Controls and More | UpGuard. https://www.upguard.com/blog/sox-compliance

U.S. Department of Health and Human Services. (n.d.-a). 45 CFR § 164.312 - Technical Safeguards. https://www.law.cornell.edu/cfr/text/45/164.312

U.S. Department of Health and Human Services. (n.d.-b). 45 CFR § 164.312 - Technical Safeguards. https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-C/part-164/subpart-C/section-164.312

U.S. Department of Health and Human Services. (n.d.-c). *Summary of the HIPAA Security Rule*. https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html

VeritasChain. (n.d.). *Building a Tamper-Evident Audit Log with SHA-256 Hash Chains (Zero Dependencies)*. DEV Community. https://dev.to/veritaschain/building-a-tamper-evident-audit-log-with-sha-256-hash-chains-zero-dependencies-h0b

y Periodismo de Investigación, R. G. (2021). Covid-19: Ecuador asignó más de US 664 millones para la pandemia y se investigan 160 casos de presunta corrupción. https://convoca.pe/investigacion/covid-19-ecuador-asigno-mas-de-us-664-millones-para-la-pandemia-y-se-investigan-160



EDITORIAL UTN
IBARRA - ECUADOR

ISBN 978-9942-572-59-2