

UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD POLITÉCNICA

INGENIERÍA EN INFORMÁTICA



Desarrollo de un sistema de monitorización y
alertas en tiempo real para la detección de
eventos críticos en la red ethereum

Denis Amilcar Giménez Alvarez
Matías Antonio Sosa Ramos

San Lorenzo - Paraguay
Noviembre - 2025

UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD POLITÉCNICA

INGENIERÍA EN INFORMÁTICA



**Desarrollo de un sistema de monitorización y
alertas en tiempo real para la detección de
eventos críticos en la red ethereum**

**Denis Amilcar Giménez Alvarez
Matías Antonio Sosa Ramos**

Asesor:

Phd. Marcos Villagra

Proyecto de Trabajo de Grado presentado en conformidad a los
requisitos para obtener el grado de Ingeniería en Informática

San Lorenzo - Paraguay
Noviembre - 2025

*Dedicamos este trabajo a nuestras familias,
por su amor incondicional y apoyo constante,
y a todas las personas que,
de una u otra forma, creyeron en nosotros.*

AGRADECIMIENTOS

Queremos expresar nuestro más sincero agradecimiento a nuestras familias y amigos, por su apoyo incondicional, su paciencia y comprensión a lo largo de todos estos años de formación académica.

Agradecemos de manera especial a nuestro tutor Marcos Villagra, por su orientación, tiempo y dedicación durante el desarrollo de este trabajo. Sus comentarios, sugerencias y confianza en nosotros fueron fundamentales para mejorar la calidad de esta investigación.

Entre nosotros, los autores: nos agradecemos mutuamente por estos veintiún años de amistad, desde la escuela hasta la culminación de esta carrera. Gracias por el apoyo, la paciencia y las incontables horas de estudio y risas compartidas.

Extendemos también nuestro agradecimiento a los docentes y compañeros de la Facultad, por los conocimientos compartidos, las discusiones técnicas y el acompañamiento en todo el proceso de aprendizaje.

Finalmente, agradecemos a las instituciones y personas que colaboraron directa o indirectamente con este proyecto, ya sea mediante el acceso a recursos, infraestructura tecnológica o intercambio de ideas que contribuyeron al resultado de este trabajo.

Desarrollo de un sistema de monitorización y alertas en tiempo real para la detección de eventos críticos en la red ethereum

Autores: Matias Sosa - Denis Giménez

Asesor: PhD Marcos Villagra

RESUMEN

Este trabajo aborda el problema de la observabilidad en Ethereum en la etapa posterior a *The Merge*, en la que la red adopta un mecanismo de consenso de Prueba de Participación y una arquitectura de doble capa que incrementa la complejidad operativa. El objetivo general consiste en diseñar y construir un sistema de monitorización y alertas en tiempo casi real que permita detectar y contextualizar eventos críticos en la red —como pérdida de *finality*, reorganizaciones de cadena o caídas de participación de validadores— a partir de los datos expuestos por la Beacon Chain API y por la capa de ejecución.

La metodología combina una revisión y sistematización del marco teórico necesario para interpretar métricas y eventos de la red con el diseño de una arquitectura de referencia basada en componentes *open source* (Apache NiFi, InfluxDB, Grafana y Nginx) desplegados mediante Docker Compose. Sobre esta arquitectura se desarrollaron artefactos de software específicos: una librería Java para consumir eventos SSE y un bundle de NiFi que integra dichos eventos en *pipelines ETL* con salida en JSON o InfluxDB Line Protocol. A partir de estos componentes se construyeron paneles de Grafana y reglas de alerta que permiten seguir en tiempo casi real la evolución de *slots* y *epochs*, la participación de validadores, la aparición de reorganizaciones y métricas clave de la capa de ejecución (gas, *blobs*).

La validación del sistema incluyó pruebas unitarias y de integración sobre los componentes desarrollados, así como ejecuciones controladas del entorno completo, observando su comportamiento ante reinicios de contenedores y fallas de red. Los resultados muestran que la arquitectura propuesta es capaz de captar y representar con fidelidad los eventos de consenso relevantes, ofreciendo una herramienta reproducible y extensible para operadores y equipos de investigación. Como contribución adicional, la publicación de la librería y del bundle de NiFi bajo licencia abierta facilita la replicabilidad del trabajo y sienta las bases para futuras extensiones hacia otros protocolos de Prueba de Participación.

Palabras Clave: Ethereum, blockchain, Apache NiFi, InfluxDB, Grafana, eventos SSE, open source.

ÍNDICE

Página

1. INTRODUCCIÓN	1
1.1. Objetivos	2
1.1.1. Objetivo general	2
1.1.2. Objetivos específicos	2
1.1.3. Metodología empleada	3
1.1.4. Organización del trabajo	3
2. Marco teórico	4
2.1. Fundamentos de blockchain	5
2.2. Arquitectura post- <i>Merge</i>	8
2.2.1. Capa de Consenso (Beacon Chain)	8
2.2.2. Capa de Ejecución	11
2.3. Tiempo en la Beacon Chain	14
2.4. Estructura mínima del bloque y execution payload	16
2.5. Modelo de gas	18
2.6. Staking y validadores	23
2.6.1. Atestaciones	23
2.6.2. Recompensas y penalizaciones	26
2.7. Eventos del <i>Beacon Node</i>	29
2.7.1. Eventos de progreso de cadena	29
2.7.2. Eventos de actividad de validadores	30
2.7.3. Eventos de seguridad (críticos)	31
2.7.4. Eventos de interfaz CL–EL y datos	32
2.7.5. Eventos de clientes ligeros y gossip	33
2.8. Salud del nodo	34
2.9. Parámetros de red relevantes	37
2.10. Síntesis operativa	40
2.11. Trabajos relacionados y soluciones de monitoreo	44
3. Sistema bajo estudio	46
3.1. Visión general del proyecto	46
3.2. Arquitectura general del sistema	47
3.3. Visión en detalle del proyecto	49
3.3.1. Fuente de Datos: Nodo Ethereum y Beacon Chain API	49
3.3.2. Arquitectura de despliegue con Docker Compose	57
3.3.3. InfluxDB – Almacenamiento de series temporales	60
3.3.4. Grafana – Visualización y análisis de métricas	62
3.3.5. Apache NiFi (Ingesta y transformación de datos)	72
3.3.6. Relación con los objetivos del trabajo	74

3.4.	Características del entorno y configuración experimental	75
3.4.1.	Plataforma de ejecución	75
3.4.2.	Clientes de la Beacon API y fuentes de datos	76
3.4.3.	Ventana de pruebas instrumentada	76
3.5.	Resultados experimentales del sistema de monitoreo	76
3.5.1.	Procedimiento experimental e instrumentación	77
4.	Herramientas desarrolladas	80
4.1.	Librería Java para Escucha de Eventos de la Beacon Chain	80
4.1.1.	Propósito	80
4.1.2.	Comparativa, estado del arte y motivación del desarrollo propio	80
4.1.3.	Objetivos y requisitos	82
4.1.4.	Alcance	83
4.1.5.	Arquitectura y diseño	83
4.1.6.	API pública y uso	84
4.1.7.	Integración con el stack	85
4.1.8.	Robustez, rendimiento y operación	86
4.1.9.	Seguridad y cumplimiento	86
4.1.10.	Validación y pruebas	87
4.1.11.	Publicación, versionado y mantenimiento	87
4.1.12.	Limitaciones y trabajo futuro	87
4.1.13.	Impacto académico y replicabilidad	87
4.2.	Controlador y procesador open-source para NiFi con el fin de recibir eventos de un nodo	88
4.2.1.	Propósito	88
4.2.2.	Motivación del desarrollo propio	88
4.2.3.	Objetivos y requisitos	89
4.2.4.	Alcance	89
4.2.5.	Arquitectura y diseño	89
4.2.6.	API pública y uso (configuración en NiFi)	90
4.2.7.	Integración con el stack	93
4.2.8.	Transformación a InfluxDB Line Protocol	93
4.2.9.	Robustez, rendimiento y operación	93
4.2.10.	Seguridad y cumplimiento	94
4.2.11.	Validación y pruebas	94
4.2.12.	Publicación, versionado y mantenimiento	95
4.2.13.	Limitaciones y trabajo futuro	95
4.2.14.	Impacto académico y replicabilidad	95
5.	CONCLUSIÓN	96

LISTA DE FIGURAS

Página

2.1. Diagrama esquemático de una cadena de bloques que muestra tres bloques consecutivos, cada uno con su marca de tiempo, nonce, hash del bloque previo y conjunto de transacciones; ilustra cómo cada bloque referencia al anterior mediante el campo Prevhash, formando una cadena enlazada criptográficamente. Fuente: [14]	5
2.2. Esquema del estado global de la cadena antes y después de la inclusión de un bloque, donde la ejecución secuencial de las transacciones t1–t3 transforma el estado world state (t) en world state (t+1). Fuente: [13]	6
2.3. Comparación esquemática entre los procesos de consenso Proof-of-Work y Proof-of-Stake, mostrando la cadena de bloques y la selección del líder en cada caso. En Proof-of-Work la elección del bloque candidato se pondera por la tasa de hash de los mineros, mientras que en Proof-of-Stake se pondera por la participación en stake de los validadores. Fuente: [15]	7
2.4. Diagrama del flujo de atestaciones en Ethereum Proof-of-Stake, donde validadores de distintos comités envían atestaciones a agregadores y nodos beacon mediante gossip, entendido como un protocolo de difusión punto a punto de mensajes entre nodos, y el bloque proponente incorpora las atestaciones agregadas para producir un nuevo bloque en la cadena. Fuente: [16]	8
2.5. Diagrama de la arquitectura en capas de un nodo de Ethereum, donde la librería Web3 se comunica con la capa de consenso (<i>consensus client</i>) y la capa de ejecución (<i>execution client</i>), que a su vez se coordinan internamente mediante la Engine API y se conectan externamente por redes p2p. Fuente: [10]	9
2.6. Diagrama del cliente de consenso de Ethereum que muestra la gestión de la Beacon Chain, los datos de blobs y el estado beacon, junto con su comunicación p2p con otros nodos. También ilustra el intercambio de mensajes RPC con el cliente de ejecución subyacente. Fuente: [17]	9
2.7. Esquema temporal de un epoch en Ethereum dividido en slots, donde en cada slot un proponente de bloque propone un nuevo bloque y un comité de validadores emite atestaciones sobre su validez. La figura ilustra la repetición de este proceso desde el inicio hasta el fin de la epoch. Fuente: [18]	10
2.8. Diagrama del cliente de ejecución de Ethereum que muestra la gestión del estado de ejecución, la EVM y el mempool de transacciones, junto con su comunicación p2p con otros nodos y el intercambio de mensajes RPC con el cliente de consenso. Fuente: [17]	11
2.9. Diagrama de la arquitectura de un nodo de Ethereum que separa la capa de consenso y la capa de ejecución dentro de un nodo local, comunicadas mediante la Engine API. La parte inferior ilustra múltiples nodos con ambas capas conectados entre sí por una red p2p para conformar la red Ethereum. Fuente: [20]	13

2.10. Esquema del tiempo hasta la finalidad en Ethereum que muestra la relación entre slots de 12 segundos, epochs de 32 slots, el rol de los checkpoints y la posibilidad de reorganizaciones donde algunos bloques confirmados pueden revertirse antes de ser finalizados. Fuente: [21]	15
2.11. Representación conceptual de la finalidad en Ethereum que distingue entre bloques finalizados y solo confirmados, indicando cómo la permanencia de las transacciones aumenta a medida que se incorporan nuevos bloques hasta el momento actual. Fuente: [21]	16
2.12. Diagrama del flujo de una transacción en Ethereum donde una cuenta externa envía un mensaje a una cuenta de contrato, se asigna un suministro de gas para ejecutar el código EVM y, tras el consumo de gas, se quema la parte utilizada y se devuelve el remanente al emisor. Fuente: [22]	19
2.13. Comparación esquemática del modelo de comisiones en Ethereum antes y después de EIP-1559, donde inicialmente todas las comisiones y la recompensa de bloque se entregan al minero y, tras la propuesta, se separan en una base fee quemada, una priority fee para el minero y la recompensa de bloque. Fuente: [23]	20
2.14. Esquema de la arquitectura de Ethereum que destaca la capa de sharding de datos basada en blobs, donde múltiples conjuntos de blobs almacenan datos de transacciones como soporte de capacidad para la ejecución en L1 y las soluciones de capa 2. Los blobs actúan como contenedores temporales de datos que alivian la carga de la EVM principal y permiten mayor escalabilidad. Fuente: [26]	22
2.15. Infografía sobre el proceso de staking en Ethereum que ilustra cómo un participante se convierte en validador al depositar 32 ETH, asume los roles de proponente y atestador de bloques y participa en comités de 128 validadores que producen y finalizan bloques a lo largo de slots y epochs.	24
2.16. Diagrama de los riesgos al operar un validador en Ethereum que clasifica las sanciones en slashing y penalizaciones menores, detallando sus causas principales (bloques conflictivos, atestaciones incorrectas e inactividad) y las consecuencias económicas y de expulsión de la red asociadas.	25
2.17. Diagrama del proceso de slashing en Ethereum que muestra cómo un validador comete una acción maliciosa, esta es detectada y reportada por un whistleblower, y el proponente incluye la evidencia en un bloque finalizado. Se detallan las penalizaciones aplicadas al validador sancionado y las recompensas otorgadas al whistleblower y al proponente del bloque.	26
2.18. Infografía sobre el estado del slashing en Ethereum a febrero de 2023, que resume el número total de validadores sancionados, la frecuencia aproximada de slashing y los principales eventos históricos asociados, incluyendo grandes incidentes y cambios en las penalizaciones. Fuente: [29]	28

3.1. Arquitectura general del sistema de monitoreo que muestra el flujo de datos desde la Beacon API hacia NiFi, su almacenamiento en InfluxDB y la visualización y notificaciones a través de Grafana.	48
3.2. Arquitectura detallada del sistema de monitoreo propuesto para Ethereum	50
3.3. Vista general de los tipos de datos expuestos por un light node de Ethereum mediante la Beacon Node API, incluyendo bloques, configuración, estado del nodo y eventos.	51
3.4. Flujo interno del procesamiento en Apache NiFi que envía métricas hacia InfluxDB para su posterior visualización en Grafana y publicación mediante NGINX.	58
3.5. Dashboard Beacon Node Status que muestra el estado operativo del nodo Ethereum Beacon.	64
3.6. Dashboard Beacon Config que muestra la configuración del protocolo y el histórico de forks de la red Ethereum.	65
3.7. Dashboard Block Metrics que muestra métricas detalladas sobre los bloques producidos en la Beacon Chain de Ethereum.	67
3.8. Dashboard Block Metrics (continuación) que muestra métricas detalladas sobre los bloques producidos en la Beacon Chain de Ethereum.	68
3.9. Canvas de Apache NiFi que muestra el flujo de ingestión y transformación de datos desde la Beacon Chain API hacia InfluxDB.	73
3.10. Canvas de Apache NiFi que muestra el flujo de ingestión y transformación de datos desde la Beacon Chain API hacia InfluxDB (continuación).	73
3.11. Canvas de Apache NiFi que muestra el flujo de ingestión y transformación de datos desde la Beacon Chain API hacia InfluxDB (continuación).	74
4.1. Arquitectura de la librería Java para la escucha de eventos de la Beacon Chain: la aplicación cliente interactúa con la interfaz <code>BeaconEventClient</code> , cuya implementación <code>OkHttpBeaconEventClient</code> gestiona las conexiones SSE, las suscripciones, la distribución de eventos y la reconexión automática utilizando <code>EventSource</code> (<code>OkHttp</code>) y <code>ObjectMapper</code> (<code>Jackson</code>) para consumir el endpoint <code>/eth/v1/events</code> del nodo de Ethereum Beacon Chain.	84
4.2. Componentes del proyecto <code>nifi-eth-events</code> : módulo de servicios, API y procesadores, cada uno empaquetado como un NAR independiente.	90
4.3. Diagrama de componentes y flujo interno: el servicio <code>StandardBeaconEventsService</code> concentra la suscripción SSE al <i>Beacon Node</i> y distribuye los eventos a múltiples <code>BeaconEventsProcessor</code> , que generan <i>FlowFiles</i> en formato JSON, Line Protocol o rutas personalizadas según la configuración.	91

LISTA DE TABLAS

Página

2.1. Comparativa resumida de soluciones de monitoreo para Ethereum.	44
3.1. Resumen de escenarios de prueba sobre el sistema de monitoreo.	78
4.1. Cuadro comparativo de listeners y alcance de eventos en Java para Ethereum. .	82

LISTA DE ALGORITMOS

Página

4.1. Suscripción a eventos de la Beacon Chain	85
---	----

LISTA DE ACRÓNIMOS Y SÍMBOLOS

Acrónimos generales

API	<i>Application Programming Interface</i> ; interfaz de programación de aplicaciones, usualmente expuesta mediante HTTP/HTTPS.
REST	<i>Representational State Transfer</i> ; estilo de diseño de APIs HTTP basado en recursos y verbos estándar.
RPC	<i>Remote Procedure Call</i> ; modelo de invocación remota utilizado, por ejemplo, en JSON-RPC para la capa de ejecución.
JSON	<i>JavaScript Object Notation</i> ; formato de intercambio de datos estructurados utilizado en las respuestas de la Beacon Chain API.
URL	<i>Uniform Resource Locator</i> ; identificador de recursos web, usado para apuntar a nodos y endpoints.
HTTP / HTTPS	<i>HyperText Transfer Protocol (Secure)</i> ; protocolos de transporte sobre los que se exponen las APIs y el canal SSE.

Ethereum y consenso

EVM	<i>Ethereum Virtual Machine</i> ; máquina virtual que ejecuta el código de contratos inteligentes en la capa de ejecución.
ETH	Unidad monetaria nativa de la red Ethereum.
nonce	Contador de transacciones asociado a cada cuenta de Ethereum, usado para ordenar envíos sucesivos y prevenir repeticiones (<i>replay</i>).
trie	Estructura de datos en árbol (Merkle Patricia trie) utilizada para organizar claves y calcular raíces como <code>state_root</code> , <code>receipts_root</code> o <code>transactions_root</code> .
PoS	<i>Proof of Stake</i> ; mecanismo de consenso de Ethereum post-Merge basado en validadores y <i>staking</i> .
PoW	<i>Proof of Work</i> ; mecanismo de consenso anterior a PoS basado en pruebas de trabajo realizadas por mineros.
CL	<i>Consensus Layer</i> ; capa de consenso de Ethereum responsable de la Beacon Chain y del protocolo PoS.
EL	<i>Execution Layer</i> ; capa de ejecución de Ethereum encargada de la EVM, transacciones y estado.
EIP	<i>Ethereum Improvement Proposal</i> ; documento que describe cambios o extensiones al protocolo de Ethereum (por ejemplo, EIP-1559, EIP-4844).

FFG	<i>Friendly Finality Gadget</i> ; esquema de finalización de Casper usado para justificar y finalizar <i>epochs</i> en Ethereum PoS.
RANDAO	Mecanismo de aleatoriedad distribuida utilizado para seleccionar proponentes y comités de validadores de forma pseudoaleatoria.
BLS	<i>Boneh–Lynn–Shacham</i> ; esquema de firma criptográfica empleado en las credenciales y firmas agregadas de validadores.
L1	<i>Layer 1</i> ; capa base de ejecución y consenso de Ethereum.
L2	<i>Layer 2</i> ; soluciones de escalabilidad construidas sobre L1 (rollups u otras).
P2P	<i>Peer-to-Peer</i> ; modelo de red entre pares utilizado por los clientes de consenso y ejecución.
NFT	<i>Non-Fungible Token</i> ; token no fungible usado como ejemplo de periodos de alta demanda de transacciones.
ZK	<i>Zero-Knowledge</i> ; hace referencia a pruebas de conocimiento cero empleadas en ciertos <i>rollups</i> de capa 2.
Streaming y monitoreo	
SSE	<i>Server-Sent Events</i> ; canal unidireccional HTTP utilizado por el endpoint <code>/eth/v1/events</code> para transmitir eventos en tiempo real.
ETL	<i>Extract, Transform, Load</i> ; patrón de flujo de datos aplicado en los pipelines de NiFi para procesar métricas y eventos.
LP	<i>Line Protocol</i> ; formato de escritura de series temporales utilizado por InfluxDB (InfluxDB Line Protocol).
Infraestructura y seguridad	
CPU	<i>Central Processing Unit</i> ; procesador principal del host donde se ejecuta el nodo, monitorizado como recurso de sistema.
NTP	<i>Network Time Protocol</i> ; protocolo utilizado para sincronizar el reloj del sistema con servidores de referencia.
TLS	<i>Transport Layer Security</i> ; protocolo criptográfico que asegura las conexiones HTTPS hacia el nodo Beacon.
Herramientas y distribución	
CLI	<i>Command-Line Interface</i> ; interfaz de línea de comandos utilizada por clientes y utilidades asociadas a Ethereum.
MIT	Licencia de software abierta <i>MIT License</i> bajo la cual se publican la librería y el bundle.

NAR

NiFi Archive; formato de empaquetado de extensiones para Apache NiFi.

CAPÍTULO 1

INTRODUCCIÓN

Ethereum se consolidó como una plataforma programable crítica para servicios financieros, logísticos y de infraestructura digital; tras la transición *The Merge*, ocurrida en 2022, opera íntegramente bajo el mecanismo de consenso de Prueba de Participación y una arquitectura dividida entre la capa de consenso (*Beacon Chain*) y la capa de ejecución [1, 2, 3]. Esta evolución redujo el consumo energético del protocolo, pero agregó complejidad operativa: los operadores deben seguir simultáneamente eventos de consenso (*slots, epochs, finality*) y métricas de ejecución (gas, blobs, transacciones), además de mantener la salud de sus nodos para evitar la pérdida de recompensas o penalizaciones. En ese contexto, los tableros de monitoreo tradicionales, centrados en contadores de la máquina virtual o en métricas genéricas de infraestructura, resultan insuficientes para explicar fenómenos propios del consenso moderno de Ethereum.

En particular, la separación entre capa de consenso y capa de ejecución hace que fenómenos como las reorganizaciones de cadena (*reorgs*) tengan una lectura más sutil. Un *reorg* se produce cuando la cadena que un nodo considera canónica es reemplazada por otra con mayor peso en términos de votos de validadores, lo que implica descartar uno o varios bloques previamente aceptados y reordenar transacciones recientes. En un entorno de Prueba de Participación bien configurado, las reorganizaciones profundas deberían ser eventos raros; su aparición suele indicar problemas de sincronización entre clientes, configuraciones sesgadas de *fork choice* o fallas operativas que amplifican riesgos económicos y de seguridad para los usuarios y operadores.

Un caso ilustrativo fue la reorganización de siete bloques consecutivos observada en la Beacon Chain durante mayo de 2022, en pleno proceso de transición hacia *The Merge*, que expuso cómo una combinación de versiones de cliente y distribuciones desbalanceadas de validadores podía generar comportamientos inesperados a escala de red. Si bien el episodio se mitigó rápidamente mediante actualizaciones de software y ajustes de parámetros, puso de manifiesto que incluso en un protocolo maduro las garantías de estabilidad dependen de supuestos operativos

frágiles: diversidad de clientes, configuraciones homogéneas y monitoreo cercano de señales de consenso. [4] Sobre este trasfondo, el diseño de Ethereum en Prueba de Participación incorpora penalizaciones económicas explícitas, conocidas como *slashing*, aplicadas a validadores que emiten mensajes contradictorios (por ejemplo, propuestas múltiples para la misma ranura o votos que se rodean entre sí) o que violan de forma verificable las reglas del protocolo. A diferencia de simples pérdidas de oportunidad por estar desconectado, los eventos de *slashing* implican la reducción permanente del saldo del validador y su expulsión gradual del conjunto activo, con penalizaciones que se amplifican cuando muchos validadores fallan de manera correlacionada. Esto convierte a la operación de nodos en una tarea de gestión de riesgo: errores de configuración, despliegues redundantes de las mismas claves o fallas de software pueden traducirse en pérdidas cuantificables si no se detectan a tiempo mediante mecanismos de monitoreo y alerta adecuados. [5]

La necesidad de observabilidad también se volvió patente por incidentes recientes, como los episodios de degradación de finality de mayo de 2023, en los que la red permaneció activa pero con validadores incapaces de finalizar bloques durante varios minutos [6]. Detectar y contextualizar a tiempo este tipo de eventos exige consumir la Beacon Node API y correlacionar flujos de datos heterogéneos [7]. Sin embargo, los conectores abiertos para dichas fuentes son escasos, muchos proyectos dejaron de mantenerse tras la incorporación de nuevas señales, y las organizaciones terminan construyendo integraciones *ad-hoc* con alto costo de mantenimiento. Además, los pipelines comerciales suelen enfocarse en telemetría de ejecución o en indicadores de infraestructura genérica, dejando un vacío para soluciones académicas, reproducibles y alineadas a prácticas modernas de ingeniería de datos.

1.1. Objetivos

1.1.1. Objetivo general

Ante este escenario, el presente trabajo tiene como objetivo general diseñar y materializar un sistema de monitoreo para Ethereum post-Merge que sea reproducible, abierto y sustentable.

1.1.2. Objetivos específicos

- Elaborar un marco conceptual actualizado sobre la operación de Ethereum en Prueba de Participación, destacando los indicadores que permiten interpretar la salud del consenso y de los nodos monitoreados.
- Diseñar una arquitectura de referencia para la ingestión, transformación, almacenamiento y visualización de eventos y métricas provenientes de la Beacon Chain, priorizando modularidad, resiliencia y seguridad.
- Desarrollar y publicar una librería *open source* capaz de suscribirse a los tópicos SSE de

la API de la beacon chain, reconectar automáticamente y exponer un modelo reutilizable en otros sistemas [8].

- Implementar un *bundle* para NiFi que procesen los eventos recibidos en formato JSON o Line Protocol, preservando tags y tipos para su posterior análisis temporal [9].
- Construir *dashboards* y reglas de alerta en Grafana basadas en series temporales almacenadas en InfluxDB, de modo que la interpretación visual respalde tanto la operación diaria como el análisis de incidencias.

1.1.3. Metodología empleada

La metodología empleada combina revisión documental de las especificaciones oficiales, diseño iterativo de arquitectura y validación experimental en entornos controlados. Se documentaron los flujos de trabajo para permitir que otros investigadores ejecuten el stack completo sin dependencias propietarias. El enfoque abierto facilita la incorporación de nuevas métricas y la adaptación del pipeline a distintas redes manteniendo la trazabilidad del proceso.

1.1.4. Organización del trabajo

El presente documento se estructura en cuatro capítulos, además de los elementos preliminares, las referencias bibliográficas y el apéndice:

- En este capítulo introductorio se presenta el contexto del problema, se motivan las necesidades de observabilidad en Ethereum post-*Merge*, se formulan los objetivos del trabajo y se expone la metodología general adoptada.
- En el Capítulo 2, *Marco teórico*, se desarrollan los conceptos fundamentales de blockchain y de la arquitectura de Ethereum en Prueba de Participación, así como las métricas y eventos de la *Beacon Chain* relevantes para el monitoreo de la red y de los nodos.
- El Capítulo 3, *Sistema bajo estudio*, describe la arquitectura de datos propuesta, el entorno experimental y las decisiones de diseño que habilitan la observabilidad planteada, junto con los resultados experimentales obtenidos al desplegar el sistema en condiciones controladas.
- El Capítulo 4, *Herramientas desarrolladas*, detalla los componentes de software implementados, su publicación como artefactos reutilizables y los experimentos de validación realizados sobre el *stack* de monitoreo.
- Finalmente, en el Capítulo 5 se sintetizan los resultados obtenidos, se discuten las limitaciones del trabajo y se proponen líneas de investigación y mejora futuras, antes de dar paso a las referencias.

CAPÍTULO 2

Marco teórico

Operar nodos de Ethereum tras *The Merge* implica comprender simultáneamente fenómenos de consenso y de ejecución y observarlos de forma sistemática. Este capítulo desarrolla el marco teórico que sustenta esa observabilidad: identifica los conceptos técnicos mínimos pero suficientes para interpretar correctamente las métricas y eventos que un sistema de monitoreo expone sobre la red y sobre un nodo en particular. El foco no está en ofrecer una enciclopedia exhaustiva de blockchain, sino en construir un vocabulario operativo que pueda leerse directamente en paneles, alertas y series temporales.

El análisis se circunscribe a Ethereum en su fase post-Merge, es decir, al protocolo vigente basado en Prueba de Participación y en la arquitectura en dos capas formada por la Beacon Chain (capa de consenso) y los clientes de ejecución [1, 10, 2]. Siempre que es posible se recurre a fuentes normativas del ecosistema y la documentación de la Beacon Node API con el objetivo de que las definiciones adoptadas coincidan con las del protocolo real [11]. Esta alineación con las especificaciones es esencial para que las métricas de monitoreo se interpreten de manera consistente entre distintos clientes y herramientas.

A partir de esa base, el capítulo avanza desde conceptos generales hacia elementos específicos del monitoreo. En primer lugar se repasan los fundamentos de blockchain necesarios para situar a Ethereum como un libro mayor distribuido y programable. Luego se describe la arquitectura post-Merge y el modelo de tiempo en la Beacon Chain [12]. Más adelante se detalla la estructura mínima de un bloque de consenso y su *execution payload*, el modelo de gas y tarifas (incluidas las modificaciones introducidas por EIP-1559 y el manejo de blobs de datos), y el funcionamiento operativo del *staking* y de los validadores, desde su activación hasta las condiciones de penalización.

Finalmente, se analizan los eventos emitidos por el cliente de consenso mediante los *end-points*, la salud del nodo y ciertos parámetros de red que condicionan la lectura de cualquier

panel de monitoreo. Al finalizar el capítulo, se busca que el lector pueda vincular cada indicador relevante (por ejemplo, una serie de eventos de reorganización, una caída de participación de validadores o un aumento sostenido de la tarifa base) con el comportamiento subyacente del protocolo y con posibles hipótesis operativas sobre el estado de la red y del nodo.

2.1. Fundamentos de blockchain

Una blockchain (cadena de bloques) es, en esencia, un libro contable distribuido que mantiene un registro inmutable de transacciones ordenadas cronológicamente. Las transacciones representan acciones que modifican el estado global del sistema (por ejemplo, transferencias de valor entre cuentas o ejecuciones de contratos inteligentes en Ethereum). Estas transacciones se agrupan en bloques; cada bloque es un paquete de múltiples transacciones que han sido validadas y aceptadas aproximadamente al mismo tiempo [13]. Los bloques son añadidos secuencialmente (ver fig. 2.1), formando una cadena donde cada bloque incluye un hash criptográfico que referencia al bloque anterior [13]. Este enlace hash entre bloques garantiza la integridad histórica: si alguien intentase alterar una transacción en un bloque pasado, el hash de ese bloque cambiaría, invalidando todos los bloques subsiguientes, lo que sería detectado por todos los nodos de la red [13]. Así, la cadena de bloques se comporta como una estructura de datos inmutable: una vez un bloque es aceptado y profundizado por suficientes bloques posteriores, su contenido (y por tanto las transacciones y estados resultantes) no puede modificarse sin rehacer todo el trabajo de cómputo y sin ser notado por la red.



Figura 2.1: Diagrama esquemático de una cadena de bloques que muestra tres bloques consecutivos, cada uno con su marca de tiempo, nonce, hash del bloque previo y conjunto de transacciones; ilustra cómo cada bloque referencia al anterior mediante el campo Prevhash, formando una cadena enlazada criptográficamente. Fuente: [14]

En Ethereum (y otras blockchains) existe la noción de estado global, que abarca todas las cuentas y contratos con sus balances, códigos y otros datos. Las transacciones son las operaciones que hacen evolucionar ese estado: por ejemplo, una transacción de transferencia de ETH disminuye el saldo del remitente y aumenta el del destinatario tal como se muestra en fig. 2.2. Para asegurar que todos los participantes acuerden el nuevo estado tras cada bloque de transacciones, es necesario un mecanismo de consenso distribuido. En la era PoW, ese consenso

se lograba mediante mineros compitiendo por resolver puzzles computacionales; en la era PoS, como veremos, se logra mediante validadores que aportan una participación (stake) en ETH y colaboran siguiendo un protocolo definido (ver fig. 2.3). En ambos casos, cada bloque propuesto debe cumplir las reglas del protocolo (transacciones válidas, referencia correcta al bloque anterior, etc.) y, tras agregarse a la cadena, su contenido es visible y verificable por cualquiera.

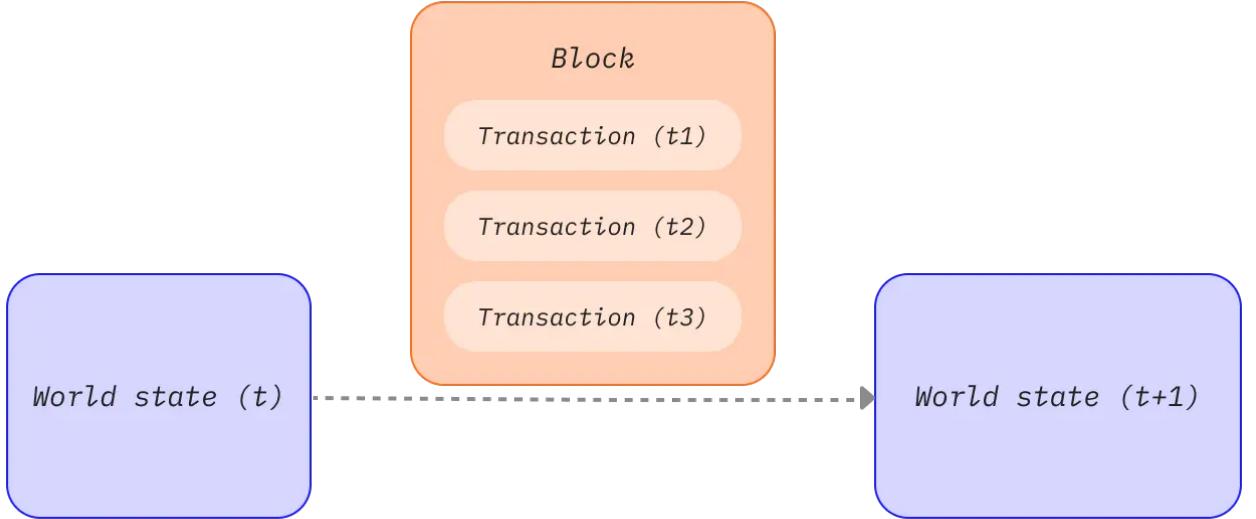


Figura 2.2: Esquema del estado global de la cadena antes y después de la inclusión de un bloque, donde la ejecución secuencial de las transacciones t1–t3 transforma el estado world state (t) en world state ($t+1$). Fuente: [13]

Un bloque contiene, además de las transacciones, una serie de metadatos importantes para la coherencia de la cadena tal como se visualiza en fig. 2.1: el hash de su bloque padre (enlace a la cadena), un sello temporal, un número de bloque o identificador de posición, y – dependiendo del sistema de consenso – otra información como pruebas de trabajo o firmas de validadores. En Ethereum post-Merge, como se detallará más adelante, cada bloque de la capa de consenso incluye un número de ramura (que indica el intervalo de tiempo fijo al que pertenece el bloque) y la identificación del validador que lo propuso [13]. Este bloque de consenso también incluye un campo especial llamado *execution payload*, que es básicamente un contenedor con las transacciones ejecutadas y el resultado de esas ejecuciones (estado resultante, logs, etc.) [13]. De esta manera, la cadena de bloques de Ethereum post-Merge combina información de consenso (orden y legitimidad de los bloques) con información de ejecución (computación de las transacciones en la EVM).

Gracias al diseño de la cadena de bloques, las propiedades de confianza y seguridad emergen de su naturaleza distribuida: todos los nodos participantes mantienen una copia del historial de bloques y ejecutan las mismas reglas de consenso para validarlos. No se requiere una autoridad central para determinar el estado “verdadero”; más bien, el protocolo asegura que, incluso si algunos actores son maliciosos o están caídos, mientras la mayoría siga las reglas, la red alcanzará acuerdo sobre una única historia de bloques (la cadena canónica). En Ethereum PoS, los nodos llegan a consenso mediante un proceso de votación (atestaciones) de los validadores

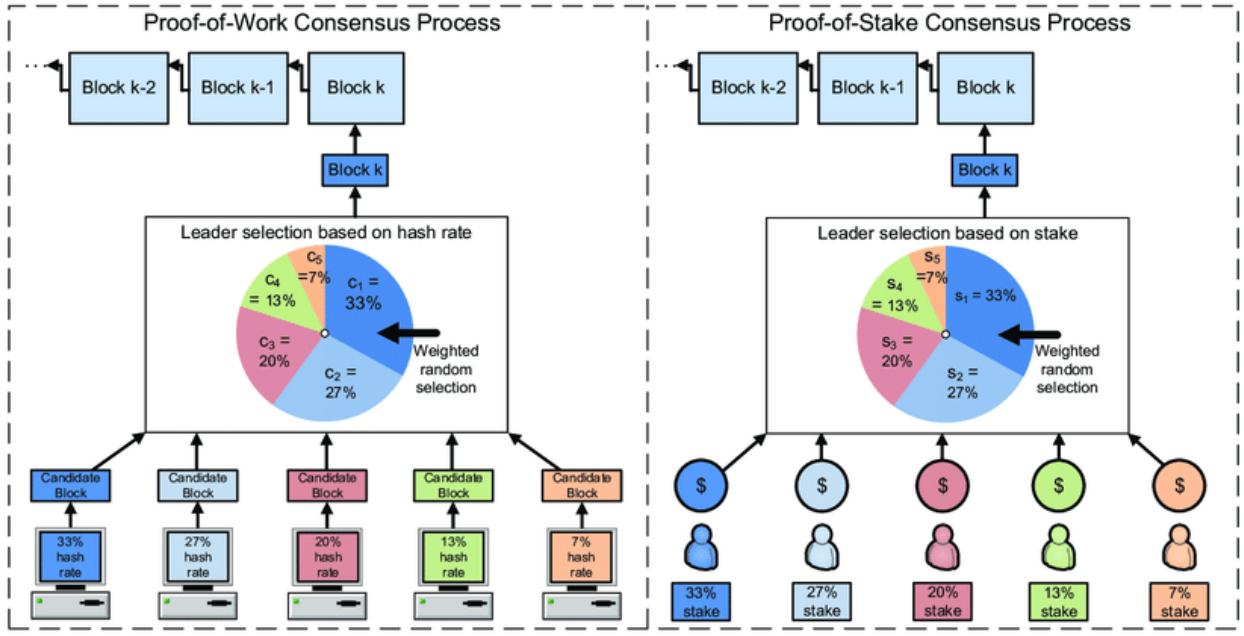


Figura 2.3: Comparación esquemática entre los procesos de consenso Proof-of-Work y Proof-of-Stake, mostrando la cadena de bloques y la selección del líder en cada caso. En Proof-of-Work la elección del bloque candidato se pondera por la tasa de hash de los mineros, mientras que en Proof-of-Stake se pondera por la participación en stake de los validadores. Fuente: [15]

(fig. 2.4), que garantiza que eventualmente los bloques se consideren finalizados (irreversibles) cuando un supermayoritario de participantes honrados los ha respaldado [2].

Este repaso rápido de los fundamentos blockchain nos prepara para entender por qué Ethereum requiere una capa de consenso compleja y bien instrumentada: solo con un consenso robusto se puede mantener la inmutabilidad y el acuerdo global sobre el historial de transacciones, características que dan valor a una blockchain pública. A medida que avancemos, profundizaremos en cómo Ethereum implementa estos principios con su arquitectura de dos capas post-Merge y cómo cada concepto básico se refleja en métricas que un sistema de monitoreo puede observar. Estos fundamentos serán aplicados posteriormente al analizar, por ejemplo, indicadores de tasa de producción de bloques, integridad de la cadena o discrepancias en el estado reportado por distintos nodos.

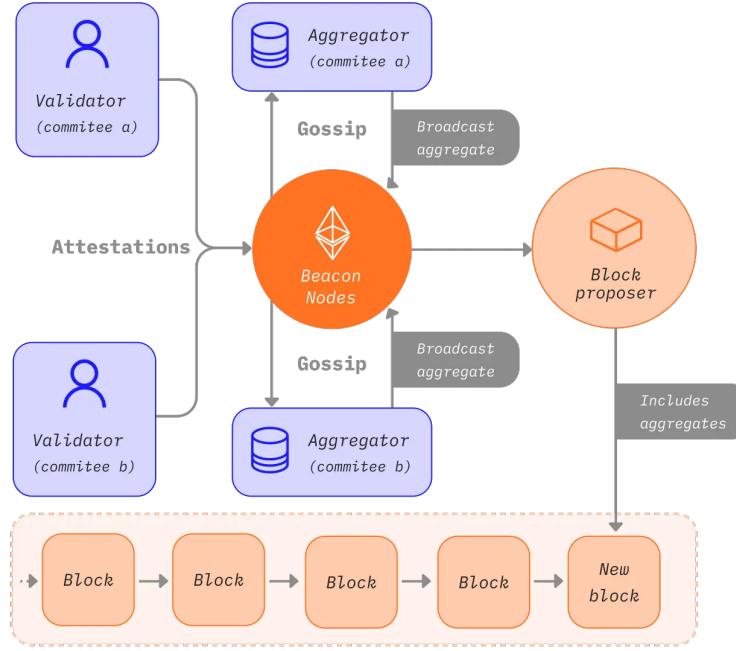


Figura 2.4: Diagrama del flujo de atestaciones en Ethereum Proof-of-Stake, donde validadores de distintos comités envían atestaciones a agregadores y nodos beacon mediante gossip, entendido como un protocolo de difusión punto a punto de mensajes entre nodos, y el bloque proponente incorpora las atestaciones agregadas para producir un nuevo bloque en la cadena. Fuente: [16]

2.2. Arquitectura post-Merge

The Merge separó claramente Ethereum en dos capas acopladas pero conceptualmente distintas (fig. 2.5): la Capa de Consenso (*Consensus Layer*, CL) y la Capa de Ejecución (*Execution Layer*, EL) [10]. Antes de *The Merge*, un solo cliente (conocido como cliente Eth1) manejaba tanto la ejecución de transacciones como el consenso PoW. Después de *The Merge*, el consenso PoS se lleva a cabo en la *Beacon Chain* (cadena Faro o de baliza), mientras que la ejecución de las transacciones y el mantenimiento del estado de la EVM sigue a cargo de los antiguos clientes de ejecución (antes llamados clientes Eth1). Ambos tipos de cliente trabajan en conjunto, comunicándose mediante una interfaz estandarizada, para dar la ilusión de un único sistema unificado [10]. Esta arquitectura modular mejora la mantenibilidad y la resiliencia: distintos equipos pueden desarrollar clientes de consenso y de ejecución en paralelo, y la red se beneficia de la diversidad de clientes sin que una sola base de código domine [10].

2.2.1. Capa de Consenso (Beacon Chain)

La Beacon Chain es responsable de coordinar el conjunto de validadores de Ethereum y de ejecutar el protocolo de consenso PoS. Cada cliente de consenso (a veces llamado Beacon Node) mantiene la lógica que decide qué cadena de bloques es la “cabeza” correcta según las reglas de *fork choice* (elección de bifurcación) y participa en la finalización de la cadena mediante el algoritmo Casper FFG (Friendly Finality Gadget).

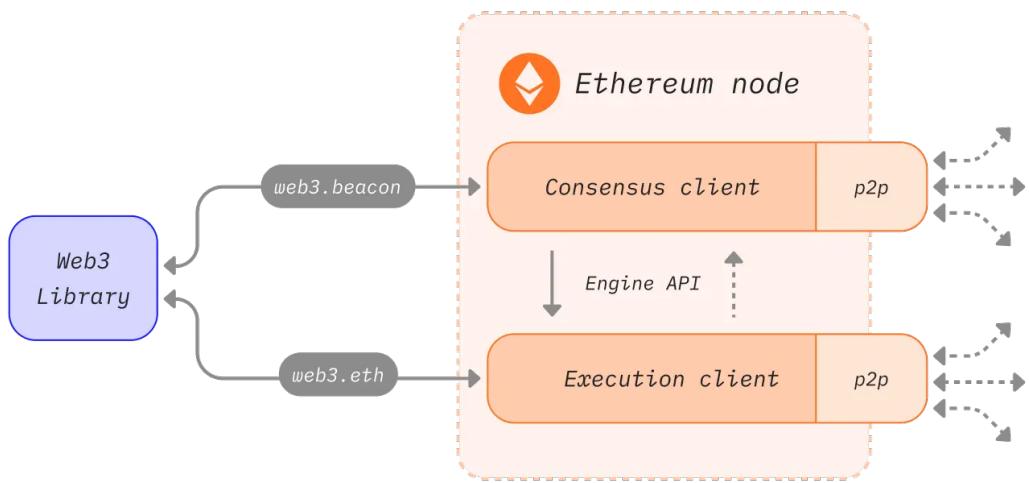


Figura 2.5: Diagrama de la arquitectura en capas de un nodo de Ethereum, donde la librería Web3 se comunica con la capa de consenso (*consensus client*) y la capa de ejecución (*execution client*), que a su vez se coordinan internamente mediante la Engine API y se conectan externamente por redes p2p. Fuente: [10]

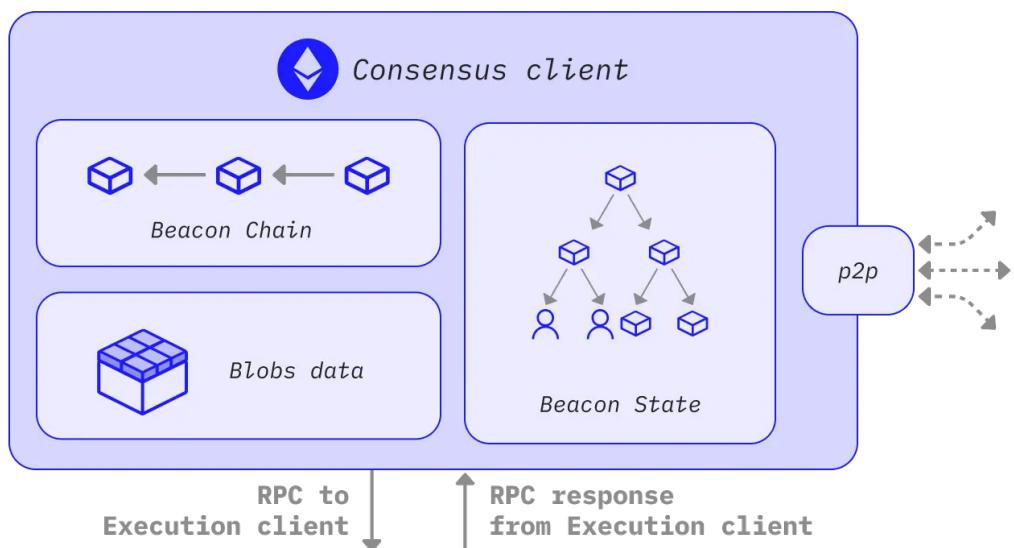


Figura 2.6: Diagrama del cliente de consenso de Ethereum que muestra la gestión de la Beacon Chain, los datos de blobs y el estado beacon, junto con su comunicación p2p con otros nodos. También ilustra el intercambio de mensajes RPC con el cliente de ejecución subyacente. Fuente: [17]

Concretamente, la Beacon Chain divide el tiempo en ranuras de 12 segundos y agrupa ranuras en épocas de 32 ranuras (fig. 2.7) [12]. En cada ranura, la capa de consenso selecciona pseudoaleatoriamente un validador para proponer un bloque de consenso, y también asigna comités de validadores que deberán atestigar (attest) sobre la validez del bloque propuesto [12]. Los bloques de la Beacon Chain contienen, además de referencias criptográficas y firmas, una serie de listas de operaciones propias del consenso: por ejemplo, listas de certificaciones (atestaciones) de validadores, *slashings* (evidencias de comportamiento indebido de validadores) y retiros de validadores (cuando están habilitados) [13]. Uno de esos componentes del bloque de consenso es el execution payload, que no es más que la porción de datos del bloque correspondiente a la capa de ejecución (ver más adelante). El cliente de consenso se comunica con su cliente de ejecución local para obtener y verificar estos payloads mediante la interfaz definida en EIP-3675/engine API [2].

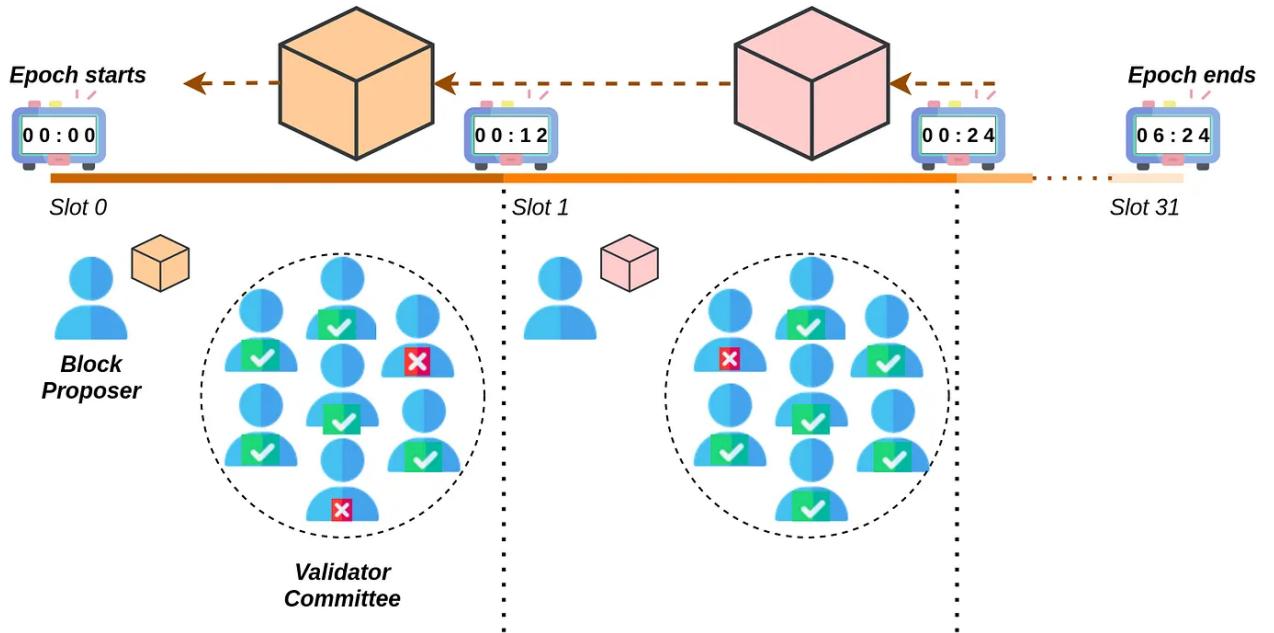


Figura 2.7: Esquema temporal de un epoch en Ethereum dividido en slots, donde en cada slot un proponente de bloque propone un nuevo bloque y un comité de validadores emite attestaciones sobre su validez. La figura ilustra la repetición de este proceso desde el inicio hasta el fin de la epoch. Fuente: [18]

Además de coordinar bloques y votos, la capa de consenso lleva la contabilidad de los validadores: registra quiénes están activos o han salido, aplica recompensas y penalizaciones según la participación de cada uno [12] [2], y procesa eventos especiales como cambios de claves de retiro o denuncias de equivocaciones (*slashing*). En resumen, el Beacon Node se encarga de coordinar el consenso: recibe bloques y attestaciones de la red P2P de consenso, verifica firmas y reglas, ejecuta el algoritmo de fork choice (LMD-Ghost) para determinar la mejor cadena, y emite eventos que informan sobre cambios de cabeza de cadena, finalización, etc. Para fines de monitoreo, la Beacon Node API expone endpoints REST que permiten consultar el estado de la capa de consenso (por ejemplo, el conjunto de validadores, los checkpoints de finalización, el estado de sincronización) y suscribirse a eventos en tiempo real (vía SSE) [19]. Muchos de

los datos que aparecerán en los paneles de monitoreo (número de ranura actual, última época finalizada, cantidad de validadores, eventos de reorg, etc.) provienen directamente de esta capa de consenso y de la información proporcionada por los clientes Beacon.

2.2.2. Capa de Ejecución

La capa de ejecución es la parte de Ethereum encargada de procesar transacciones en la EVM, aplicar los cambios de estado y gestionar la base de datos de estado (saldos de cuentas, almacenamiento de contratos, etc.). Un cliente de ejecución (también llamado Execution Engine o cliente Eth1) se ocupa de mantener la Blockchain de Ejecución, que post-Merge es esencialmente una secuencia de payloads de ejecución contenidos dentro de los bloques de la Beacon Chain. Desde la perspectiva de un cliente de ejecución, cada nuevo bloque consiste en un lote de transacciones a ejecutar, más información adicional como la dirección del beneficiario de las tarifas (ahora el validador proponente en lugar de un minero), la tarifa base, raíces de Merkle para validar el estado y logs, etc. [13]. El cliente de ejecución expone la conocida API JSON-RPC (y otras interfaces similares) para que usuarios y aplicaciones puedan enviar transacciones. Cuando un usuario o contrato origina una transacción, típicamente la envía a través de un nodo de ejecución utilizando esta API; el cliente de ejecución valida la transacción (firma correcta, cuenta con fondos suficientes, nonce adecuado) [2] y, si es válida, la coloca en su mempool (zona de espera de transacciones pendientes) [2]. Constantemente, el cliente de ejecución comparte las transacciones de su mempool con sus pares (otros clientes de ejecución) en la red P2P de la capa de ejecución, de modo que las transacciones se difunden ampliamente.

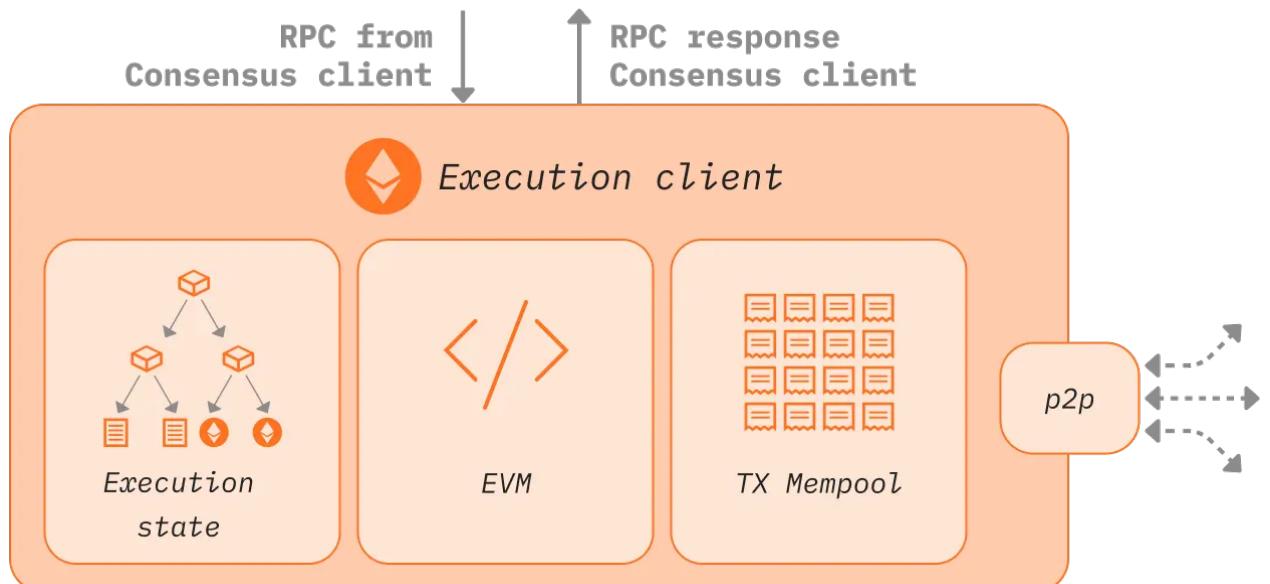


Figura 2.8: Diagrama del cliente de ejecución de Ethereum que muestra la gestión del estado de ejecución, la EVM y el mempool de transacciones, junto con su comunicación p2p con otros nodos y el intercambio de mensajes RPC con el cliente de consenso. Fuente: [17]

En el momento en que un bloque debe ser propuesto (cada 12 segundos en una ranura dada),

el cliente de consenso solicitará a su cliente de ejecución que ensamble un nuevo *execution payload* con las transacciones pendientes [2]. El *Execution Engine* entonces selecciona un conjunto de transacciones (normalmente priorizando por tarifas ofrecidas) que quepan en el límite de gas del bloque, ejecuta cada transacción secuencialmente en la EVM, y calcula el nuevo estado resultante. Este proceso genera todos los resultados necesarios del bloque: la lista ordenada de transacciones incluidas, la raíz del trie de recibos de transacción (*receipts root*), la raíz de estado final tras aplicar el bloque (*state root*) y otros campos de la cabecera, como el *block hash* del payload y el *logs bloom*, entre otros [13]. Toda esa información conforma el Execution Payload, que es devuelto al cliente de consenso. El cliente de consenso entonces inserta ese payload en el cuerpo del nuevo bloque de la Beacon Chain que está construyendo [2], junto con las atestaciones y demás datos de consenso, y finalmente firma y publica el bloque completo en la red de consenso para que otros validadores lo validen. Cuando los otros nodos (consenso+ejecución) reciben el nuevo bloque a través del gossip de la Beacon Chain, pasan el payload de ejecución a sus clientes de ejecución locales, los cuales re-ejecutan todas las transacciones de ese payload para verificar que la transición de estado es válida y coincide con el state root declarado [2]. Solo si el Execution Payload es válido, el cliente de consenso considerará válido el bloque; luego los validadores emitirán sus votos (atestaciones) afirmando que aceptan ese bloque como la cabeza de la cadena (o lo rechazarán si hubiese sido inválido por discrepancias).

Este modelo de doble cliente (fig. 2.9) implica que para monitorear correctamente un nodo Ethereum, se deben vigilar tanto aspectos de la capa de consenso como de la capa de ejecución. La Capa de Consenso aporta métricas sobre finalidad, participación de validadores, forks y reorganizaciones, mientras que la Capa de Ejecución aporta métricas sobre rendimiento de las transacciones, uso de gas, tiempos de ejecución, estado de la EVM, etc. Por ejemplo, un panel de monitoreo podría correlacionar la tasa de producción de bloques (dato de la CL) con el consumo de gas por bloque y la tarifa base en esos bloques (datos de la EL) para evaluar congestión. Es importante recalcar que ambas capas están interconectadas: un fallo en la capa de ejecución (por ejemplo, si el Execution Engine se cuelga o entra en error) hará que el cliente de consenso entre en modo optimista (seguirá proponiendo y recibiendo bloques sin validar la ejecución, se hablará de esto en la sección 2.8) y eventualmente marcará el nodo como no confiable. Del mismo modo, problemas en la capa de consenso (por ejemplo, caída de peers de consenso) impedirán al cliente de ejecución recibir nuevas transacciones confirmadas. Los sistemas de monitoreo deberán, por tanto, observar las dos capas. En secciones posteriores, esta separación de responsabilidades nos permitirá interpretar qué paneles corresponden a datos de consenso (ej. número de epoch finalizado, cantidad de atestaciones) y cuáles a datos de ejecución (ej. uso de gas, tamaño del mempool), asegurando una visión integral del estado del nodo y de la red.

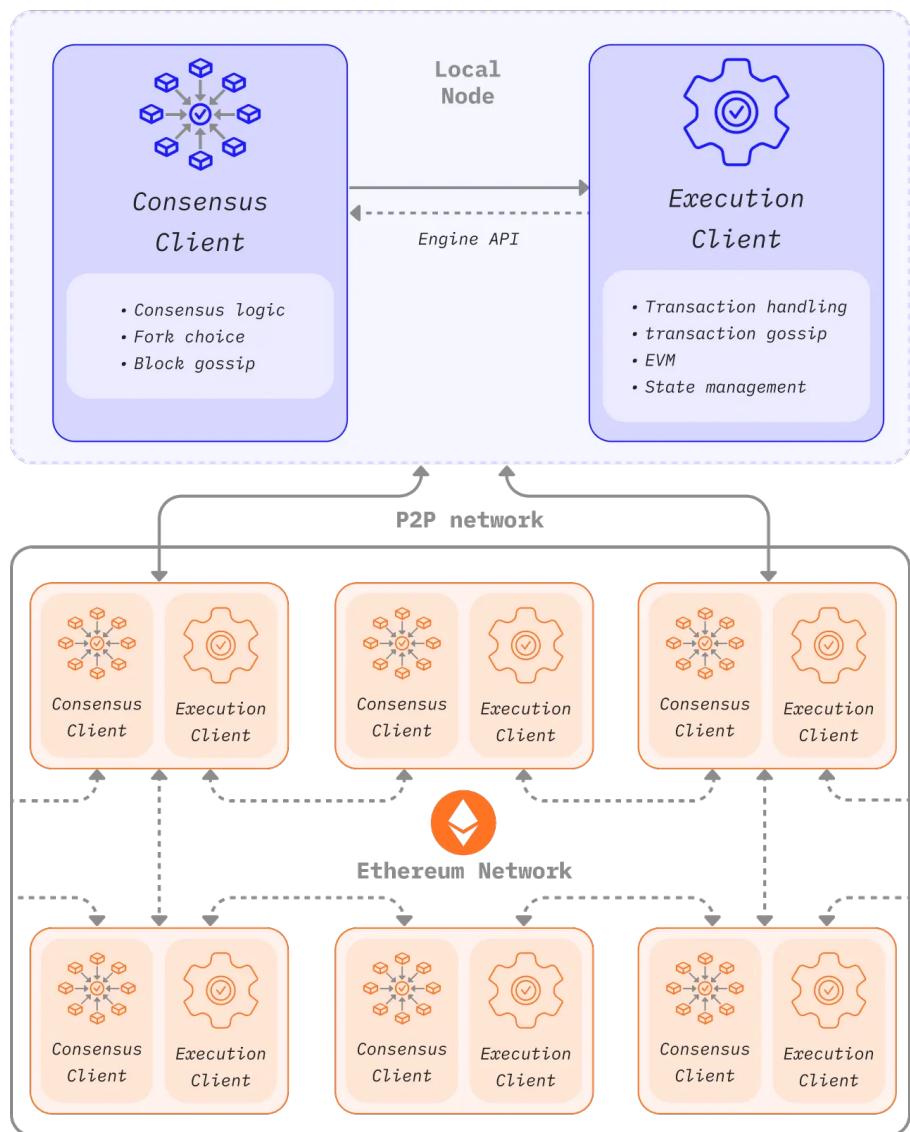


Figura 2.9: Diagrama de la arquitectura de un nodo de Ethereum que separa la capa de consenso y la capa de ejecución dentro de un nodo local, comunicadas mediante la Engine API. La parte inferior ilustra múltiples nodos con ambas capas conectados entre sí por una red p2p para conformar la red Ethereum. Fuente: [20]

2.3. Tiempo en la Beacon Chain

Ethereum bajo PoS introduce un “reloj” interno muy preciso para el consenso: el tiempo se divide en ranuras (*slots*) de duración fija (12 segundos cada una) y en épocas (*epochs*) que comprenden 32 ranuras (es decir, 6,4 minutos como se detalla en fig. 2.10) [12]. Este ritmo fijo marca una diferencia con las blockchains PoW, donde los intervalos de bloque son probabilísticos. En la Beacon Chain, cada ranura de 12 segundos es una oportunidad para proponer un bloque: idealmente, se produce exactamente un bloque por ranura. Si el validador designado para esa ranura está en línea y cumple su tarea, tendremos un nuevo bloque; si está fuera de línea o falla, la ranura pasa vacía (sin bloque) [13]. En cualquier caso, el reloj avanza imperturbable de ranura en ranura. Las épocas sirven como una unidad lógica superior: cada epoch de 32 slots permite agrupar votaciones de los validadores y definir hitos de consenso llamados checkpoints (puntos de control). El primer bloque (si existe) de cada epoch se considera un checkpoint, que luego será utilizado en el proceso de justificación y finalización (finality).

Dentro de cada epoch, todos los validadores activos participan al menos una vez votando mediante atestaciones (también llamadas certificaciones, como se puede ver en fig. 2.7). Los validadores se dividen aleatoriamente en comités distribuidos a lo largo de las ranuras de la epoch, de manera que en cada ranura solo atesta un subconjunto del total [12]. Esto mantiene la carga de comunicación manejable. Al final de la epoch, se habrá reunido el voto (atestación) de cada validador acerca de la cadena que considera válida hasta ese punto. Esos votos tienen dos propósitos: uno inmediato, ayudar a elegir la cabeza de la cadena correcta (el bloque head más reciente) según el algoritmo LMD-Ghost, y otro de más largo plazo, que es votar por los checkpoints de epoch para alcanzar la finalización.

Es crucial distinguir entre la cabeza de la cadena (head) y la cadena finalizada. La cabeza se refiere al último bloque válido en la rama que el nodo considera actualmente la mejor (esencialmente el tip de la blockchain en curso). Debido a la naturaleza distribuida y asíncrona de la red, es posible que se produzcan pequeñas bifurcaciones temporales: por ejemplo, dos validadores podrían proponer bloques distintos en la misma ranura (debido a retrasos de red), creando dos ramas competidoras. El protocolo de consenso especifica cómo los nodos eligen una sobre otra usando las atestaciones (fork choice rule), pero antes de resolverse, la cabeza puede cambiar, en eventos conocidos como reorganizaciones de cadena (reorgs). Estos reorgs suelen ser de corto alcance (uno o dos bloques) y rápidamente la red converge en una sola cabeza. Así, la cadena head es la visión más reciente pero puede sufrir reverisiones pequeñas.

Por su parte, la finalidad (finality) es una propiedad más fuerte: cuando una bloque queda finalizado, se vuelve extremadamente difícil y costoso revertirlo (fig. 2.11) . Ethereum PoS logra la finalización mediante el proceso de justificación y finalización de checkpoints: al terminar cada epoch, los validadores han votado por un par de checkpoints (source y target). Si más de dos tercios del total de ETH en staking votan de forma consistente por un checkpoint como target, ese checkpoint se marca como justificado. Y si el checkpoint inmediatamente siguiente logra justificarse, el anterior pasa a ser finalizado [2]. En términos simples, se requiere que $\geq 2/3$ de

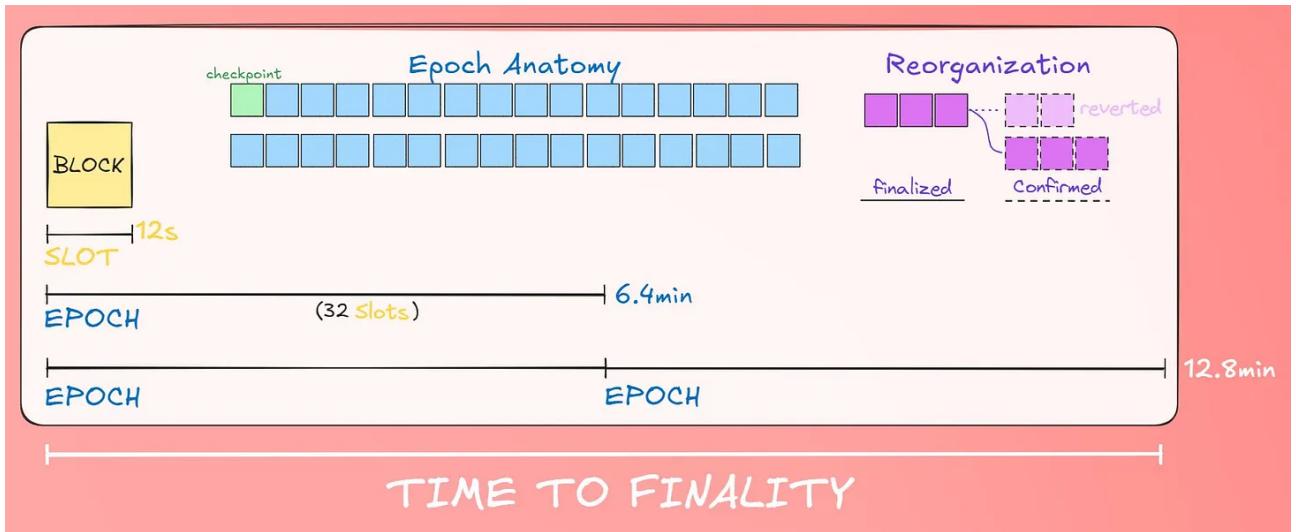


Figura 2.10: Esquema del tiempo hasta la finalidad en Ethereum que muestra la relación entre slots de 12 segundos, epochs de 32 slots, el rol de los checkpoints y la posibilidad de reorganizaciones donde algunos bloques confirmados pueden revertirse antes de ser finalizados. Fuente: [21]

los validadores apoyen una historia consistente durante dos epochs consecutivas para finalizar el epoch anterior. Una vez finalizado, un bloque y todos sus predecesores no pueden cambiarse sin que al menos un tercio de los validadores honestos sean penalizados drásticamente (perdiendo su stake) [2]. Revertir un bloque finalizado implicaría un ataque muy impracticable, ya que un atacante debería sacrificar al menos $\approx 1/3$ del total de ETH en stake, lo cual económicamente desincentiva cualquier intento [2].

La Beacon Chain implementa además un mecanismo llamado *inactivity leak* (fuga por inactividad) que actúa como red de seguridad para la finalización: si la red deja de finalizar (por ejemplo, porque justo 1/3 de los validadores dejaron de participar, bloqueando el quórum necesario), entonces gradualmente se reduce el balance (stake efectivo) de los validadores inactivos, hasta que el porcentaje de activos supera de nuevo 2/3 y se pueda finalizar [2]. Este leak comienza tras 4 epochs sin finalización (≈ 25.5 minutos) y garantiza que incluso si justo un tercio está fuera (o es malicioso votando en contra), eventualmente serán penalizados hasta restablecer la supermayoría honesta [2]. En la práctica, la finalización es un indicador de salud de la red: una Ethereum sana finaliza cada epoch (cada ≈ 6.4 minutos) sin problemas; si la finalización se detiene, es señal de un problema grave (p.ej., una gran cantidad de validadores caídos o algún error de consenso).

En términos de monitoreo, esta distinción entre la cabeza y el estado finalizado es fundamental. Habrá métricas y paneles que muestren datos basados en la cabeza de la cadena (por ejemplo, “slot actual del head”, “número de bloque head”, “tasa de proposición de bloques”), las cuales reflejan el estado inmediato y de vanguardia de la red, pero potencialmente sujeto a cambios menores. Otras métricas estarán basadas en la finalidad (por ejemplo, “último epoch finalizado”, “tiempo desde la última finalización”), las cuales ofrecen certeza de irreversibilidad. Un analista debe saber interpretarlas: por ejemplo, una discrepancia entre el head y el último

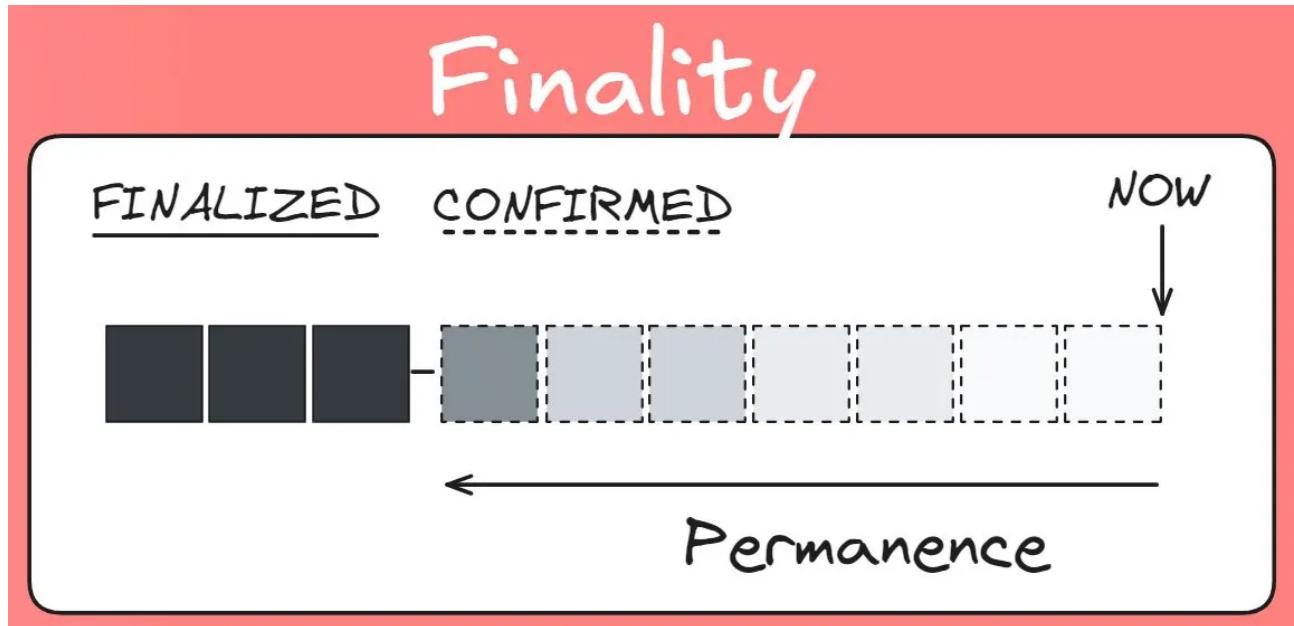


Figura 2.11: Representación conceptual de la finalidad en Ethereum que distingue entre bloques finalizados y solo confirmados, indicando cómo la permanencia de las transacciones aumenta a medida que se incorporan nuevos bloques hasta el momento actual. Fuente: [21]

finalizado es normal (el head avanza constantemente cada 12 s, mientras un punto finalizado se establece cada ≈ 2 epochs). Sin embargo, si observa que han pasado más de unos pocos minutos sin que un nuevo bloque sea finalizado, esto es un síntoma alarmante de que la red no está alcanzando consenso firme (posiblemente necesitando acción). Asimismo, para ciertas mediciones (como la tasa de participación de validadores), es común distinguir entre participación en la cabeza vs en la finalización. Este marco temporal de slots y epochs, con finalización diferida, nos permitirá luego entender paneles que muestren, por ejemplo, “distancia en slots hasta el head” versus “distancia en epochs hasta finality” para evaluar cuán actualizado está un nodo. En secciones posteriores, estos conceptos de tiempo se aplicarán al analizar métricas de bloques (por ranura), selección de cabeza de cadena y consecución de puntos finalizados, permitiendo juzgar la consistencia y estabilidad de la cadena monitorizada.

2.4. Estructura mínima del bloque y execution payload

Un bloque de Ethereum post-Merge consta de dos niveles de estructura: la parte de consenso (propia de la Beacon Chain) y la parte de ejecución (el payload de la capa de ejecución). Entender al menos los campos principales de un bloque nos permitirá luego correlacionar eventos y métricas entre ambas capas.

Campos de consenso en un bloque Beacon: Cada bloque de la Beacon Chain contiene encabezados que incluyen: el número de ranura al que corresponde el bloque, el índice del validador proponente (que identifica cuál de los validadores activos creó el bloque) [13], el hash del parent_root, que vincula al bloque anterior de la cadena) [13], y el hash de estado (state_root,

una raíz Merkle que representa el estado interno de la Beacon Chain tras procesar este bloque) [13]. Adicionalmente, viene la sección del cuerpo del bloque, la cual agrupa varias listas de objetos de consenso importantes [13]:

- Lista de proposer_slashings: evidencia de validadores que propusieron dos bloques conflictivos en la misma ranura (un acto deshonesto que conlleva slashing).
- Lista de attester_slashings: evidencia de validadores que emitieron atestaciones contradictorias (otro acto penalizable con slashing).
- Lista de attestations: las atestaciones (votos) que los distintos comités de validadores han hecho en las últimas ranuras, recopiladas para incluirse en este bloque [13]. Cada objeto de atestación incluye los datos del voto (que contienen, entre otros, el hash del bloque que atestiguan como cabeza, y los checkpoints fuente y destino que apoyan para finalización [13]), un mapa de bits indicando qué validadores del comité participaron y una firma agregada del comité [13].
- Lista de deposits: nuevos depósitos de ETH provenientes del contrato de depósito de la cadena de ejecución, que activan validadores (solo relevantes en la fase inicial, cuando ingresan validadores desde la Eth1 a la Beacon Chain).
- Lista de voluntary_exits: indicaciones de validadores que voluntariamente están saliendo del conjunto activo (solicitudes de retiro).
- Sync_aggregate: información resumida de firmas del comité de sincronización (usadas para clientes ligeros).
- Y crucialmente, el campo execution_payload: la carga útil de ejecución que provee todas las transacciones y resultados de la capa de ejecución para este bloque [13].

Esta enumeración no es exhaustiva (omitiendo algunos campos técnicos como randao_reveal, eth1_data, graffiti, etc., que aunque tienen su importancia – por ejemplo, randao_reveal contribuye a la aleatoriedad del sorteo de validadores [13] – no suelen ser foco de métricas de alto nivel). Para fines de monitoreo conceptual, interesa saber que en cada bloque de la Beacon Chain queda anclada una sección entera correspondiente a la actividad de la Capa de Ejecución. Ese anclaje es el execution_payload y su encabezado asociado. Significa que por cada bloque de consenso (cuando la ranura no está vacía), existe un bloque de ejecución “empaquetado” dentro. De hecho, la Blockchain de Ejecución (en términos de número de bloque, hashes, etc.) avanza sincronizada con cada nuevo bloque de consenso que tenga un payload válido. Por eso, tras The Merge, hablamos de una única cadena donde los bloques vienen de la Beacon Chain, a diferencia de antes donde existía una cadena Eth1 separada; ahora el progreso de bloques de ejecución está supeditado a la Beacon Chain y no hay minería independiente produciendo bloques.

El Execution Payload en sí mismo tiene su propia estructura, muy parecida a la de un bloque Ethereum PoW clásico (Bloque de Ethereum 1.0). Comprende: el hash del padre de ejecución (parent_hash del bloque de ejecución anterior) [13], la dirección del beneficiario de las tarifas (fee_recipient, que antes era el minero y ahora es el validador que propone el bloque) [13], el state_root de la EVM tras aplicar las transacciones de este bloque [13], el receipts_root (raíz del trie de recibos de transacción) [13], el logs_bloom (un filtro Bloom agregado de todos los logs de eventos generados en transacciones, útil para búsquedas) [13], el prev_randao (un valor aleatorio heredado del consenso, antiguamente el campo mix digest de PoW, ahora derivado de RANDAO en PoS) [13], el número de bloque (block_number en la cadena de ejecución) [13], el gas_limit y gas_used en este bloque [13], la timestamp (que coincide esencialmente con la hora de la ranura multiplicada por 12s) [13], el extra_data (campo libre para el proponente, heredado de PoW, a veces usado por pools para etiquetas) [13], la base_fee_per_gas (tarifa base vigente para este bloque, introducida por EIP-1559) [13], el block_hash de este bloque de ejecución (hash calculado sobre los campos de la cabecera de ejecución) [13], el transactions_root (raíz del trie de transacciones incluidas) [13] y el withdrawals_root (raíz de los retiros, campo añadido en Capella por EIP-4895) [13].

La diferencia entre el header del payload y el payload completo es que el header solo tiene las raíces de las estructuras, mientras que el payload completo contiene la lista explícita de transactions incluidas y, si aplica, la lista de withdrawals procesados en ese bloque [13]. Tras Shanghai/Capella, los retiros de validadores salen de la capa de consenso hacia la de ejecución como una lista de operaciones especiales (ver sección 2.6), cuya presencia se refleja en esos campos de withdrawals.

2.5. Modelo de gas

Ethereum utiliza el concepto de gas como unidad de medida del costo computacional de las transacciones y las operaciones en la EVM [22]. Cada instrucción de la máquina virtual y cada recurso consumido (escritura de almacenamiento, lectura de logs, bytes de datos enviados, etc.) tiene un costo en gas. Al ejecutar una transacción, el cliente de ejecución va descontando gas del límite asignado a la transacción por el remitente. El propósito de este mecanismo es doble: (1) prevenir la ejecución de cálculos excesivamente largos o infinitos (ya que el gas es finito, si se agota la ejecución se detiene) [22], y (2) establecer un mercado de tarifas donde los usuarios pagan por los recursos que sus transacciones consumen, incentivando el uso eficiente de la red (fig. 2.12) .

Gas limit vs gas used (capacidad vs uso): Cada bloque de ejecución tiene un límite de gas (gas limit), que determina la capacidad máxima de cómputo y espacio de transacciones que se puede incluir en ese bloque [13]. Históricamente, este límite era un parámetro ajustable por los mineros (y ahora por los validadores) voto a voto, pero con restricciones: puede subir o bajar a lo sumo un 0,1% por bloque, para evitar cambios bruscos [13]. Desde London (EIP-1559), el

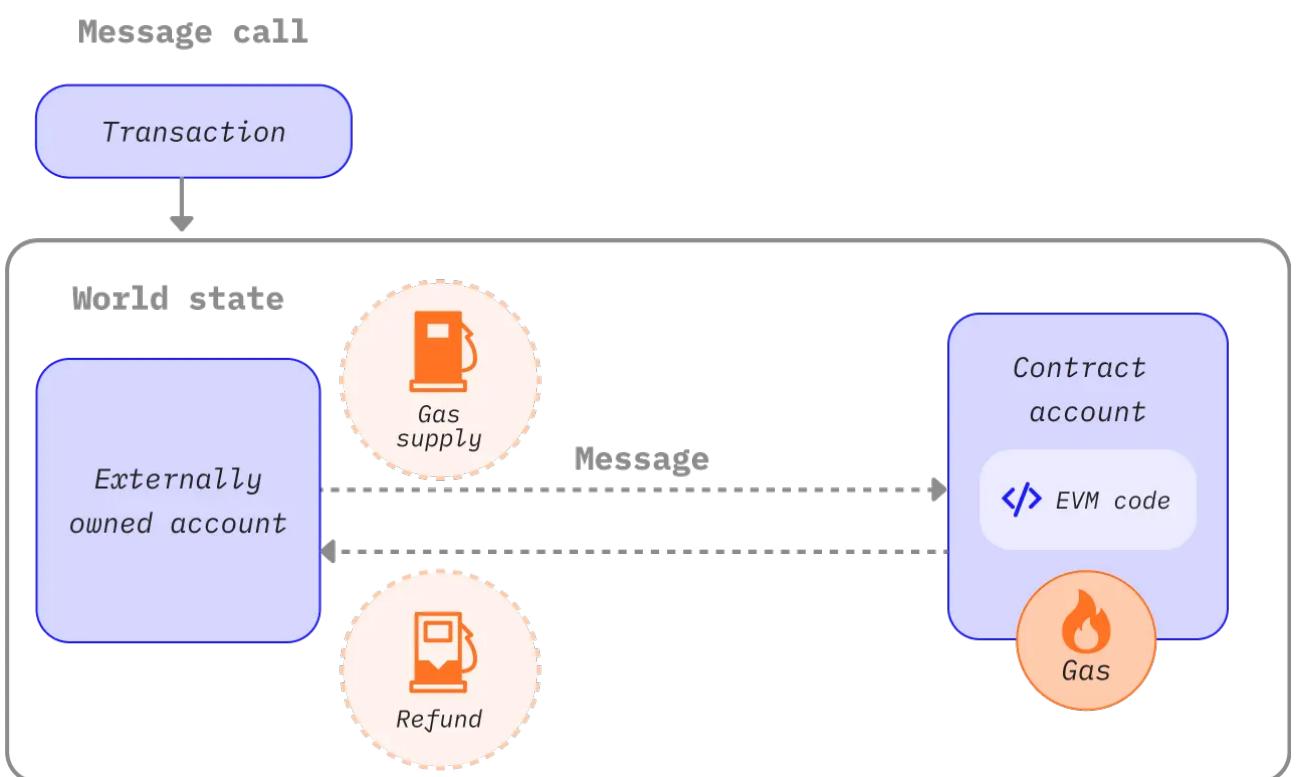


Figura 2.12: Diagrama del flujo de una transacción en Ethereum donde una cuenta externa envía un mensaje a una cuenta de contrato, se asigna un suministro de gas para ejecutar el código EVM y, tras el consumo de gas, se quema la parte utilizada y se devuelve el remanente al emisor. Fuente: [22]

protocolo define un tamaño de bloque objetivo equivalente a la mitad del gas limit máximo; por ejemplo, si el gas limit es 30 millones, el objetivo es 15 millones de gas por bloque [13]. En la realidad, los bloques pueden temporalmente ser más grandes que el objetivo (hasta el límite completo) o más pequeños, dependiendo de la demanda de transacciones. El gas used es la suma de gas efectivamente consumido por todas las transacciones incluidas en el bloque [13]. Una métrica frecuente de monitoreo es el porcentaje de utilización del bloque = $\text{gas_used} / \text{gas_limit}$. Valores consistentemente altos (cercaos al 100 % del límite) indican que la demanda de espacio en bloques es alta – es decir, hay muchas transacciones queriendo entrar, llenando los bloques. Valores bajos indican bloques poco llenos (baja demanda en ese momento).

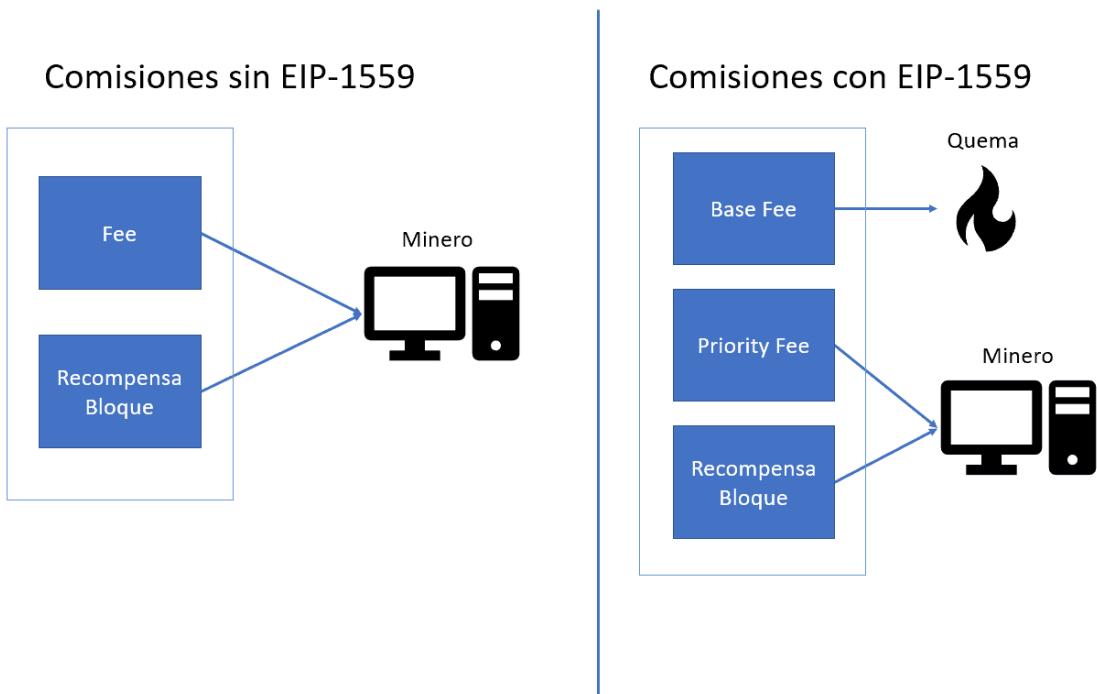


Figura 2.13: Comparación esquemática del modelo de comisiones en Ethereum antes y después de EIP-1559, donde inicialmente todas las comisiones y la recompensa de bloque se entregan al minero y, tras la propuesta, se separan en una base fee quemada, una priority fee para el minero y la recompensa de bloque. Fuente: [23]

Para los usuarios, EIP-1559 significa que al enviar una transacción ya no necesitan adivinar exactamente cuánto pagar; típicamente aceptan la base fee actual y añaden una propina al validador llamada tarifa de prioridad (priority fee, antes “tip”) [22]. En la transacción se especifican dos valores: `max_fee_per_gas` (la tarifa total máxima dispuesta a pagar por unidad de gas) y `max_priority_fee_per_gas` (la parte de esa tarifa destinada al validador). El protocolo se encarga de que, en la inclusión, el usuario pague exactamente `base_fee + priority_fee` (hasta los máximos, devolviendo el excedente si `max_fee` era mayor) [11]. Así, el coste para el usuario se divide en: `base fee` (quemada) + `propina` (para el validador). Un efecto práctico observable tras EIP-1559 es que las tarifas gas se volvieron más predecibles; en períodos de congestión, la `base fee` sube rápidamente de un bloque a otro, equilibrando la demanda, en lugar de dispararse caóticamente como antes [11].

En paneles de monitoreo, la base fee es un indicador directo de la presión de demanda sobre Ethereum. Por ejemplo, durante un NFT drop o un periodo de intensa actividad DeFi, muchas transacciones compiten por espacio: los bloques irán al máximo ($\text{gas_used} \approx \text{gas_limit}$) y esto causará incrementos de base fee bloque a bloque. Gráficamente se verá la base fee subir (a veces a cientos de gwei) y luego, cuando la demanda baja, ir descendiendo exponencialmente. Monitorear la base fee puede servir para gatillar alertas de congestión de la red o para evaluar costos de uso. Asimismo, la métrica de gas usado por bloque comparada con el gas target (punto medio) puede mostrar si la red está en persistente saturación (bloques casi dobles del target, base fee en ascenso) o en estado relajado (bloques a medio llenar, base fee bajando hacia mínimos). Los conceptos de EIP-1559 serán aplicados luego al analizar paneles de rendimiento: por ejemplo, entenderemos que un alto porcentaje de bloques con $\text{gas_used} = \text{gas_limit}$ implica base fee aumentando, lo cual se reflejará en gráficos de tarifas quemadas por minuto.

EIP-4844 y blob gas (datos fuera de la EVM): Ethereum está en proceso de introducir nuevos tipos de transacciones para escalar el throughput, en particular mediante Proto-Danksharding (EIP-4844). EIP-4844 agrega las llamadas blob-carrying transactions, transacciones que llevan consigo blobs de datos de tamaño grande (por ejemplo ≈ 128 KB cada uno) que no son accesibles por la EVM [24] pero que se almacenan temporalmente en los nodos para aumentar la disponibilidad de datos para las capas 2 (rollups) [25]. Estos blobs permiten que las soluciones de escalado (rollups optimistas o ZK) publiquen sus datos de prueba o transacciones fuera del costoso campo calldata, haciéndolo mucho más barato. La contrapartida es que los blobs no permanecen para siempre en la cadena: se mantienen solo por un período (por ejemplo, ≈ 18 días o 4096 epochs [25]) y luego pueden ser eliminados, dado que su fin es servir de buffer de datos recientes para los clientes de capa 2, no un historial permanente.

Desde el punto de vista del monitoreo, tras activarse EIP-4844 (previsto en la actualización Deneb/Cancun), aparecerán nuevas métricas relacionadas a blobs y blob gas. Por ejemplo, se medirá cuántos blobs por bloque se están incluyendo, el porcentaje de uso del espacio de blobs (similar a $\text{gas_used}/\text{gas_limit}$ pero para $\text{blob_gas_used}/\text{blob_gas_limit}$), y la base fee de blob gas. En períodos donde muchas L2s quieran publicar datos (ej: mucho volumen de rollups), podríamos ver bloques alcanzando el máximo de 6 blobs, y la base fee de blob gas incrementándose, encareciendo temporalmente ese recurso [27]. Esto es intencional: mantiene bajo control el uso de datos para que no sobrecargue a los nodos. Así, los paneles de monitoreo de una Ethereum post-EIP-4844 deben incorporar esta dimensión adicional. Un posible panel podría mostrar “uso de gas vs uso de blob gas por bloque” para ver la ocupación de ambas dimensiones. Otro podría graficar la base_fee_per_gas y la $\text{base_fee_per_blob_gas}$ a lo largo del tiempo, para ver cómo cada una responde a la demanda en su respectivo mercado [24]. En la práctica, entender que gas y blob gas son recursos separados nos previene de confusiones: podríamos tener una situación donde los bloques aún tienen espacio de gas de ejecución libre (pocas transacciones normales) pero están llenos de blobs, o viceversa. Para evaluar la capacidad y carga de la red integralmente, habrá que considerar ambas métricas.

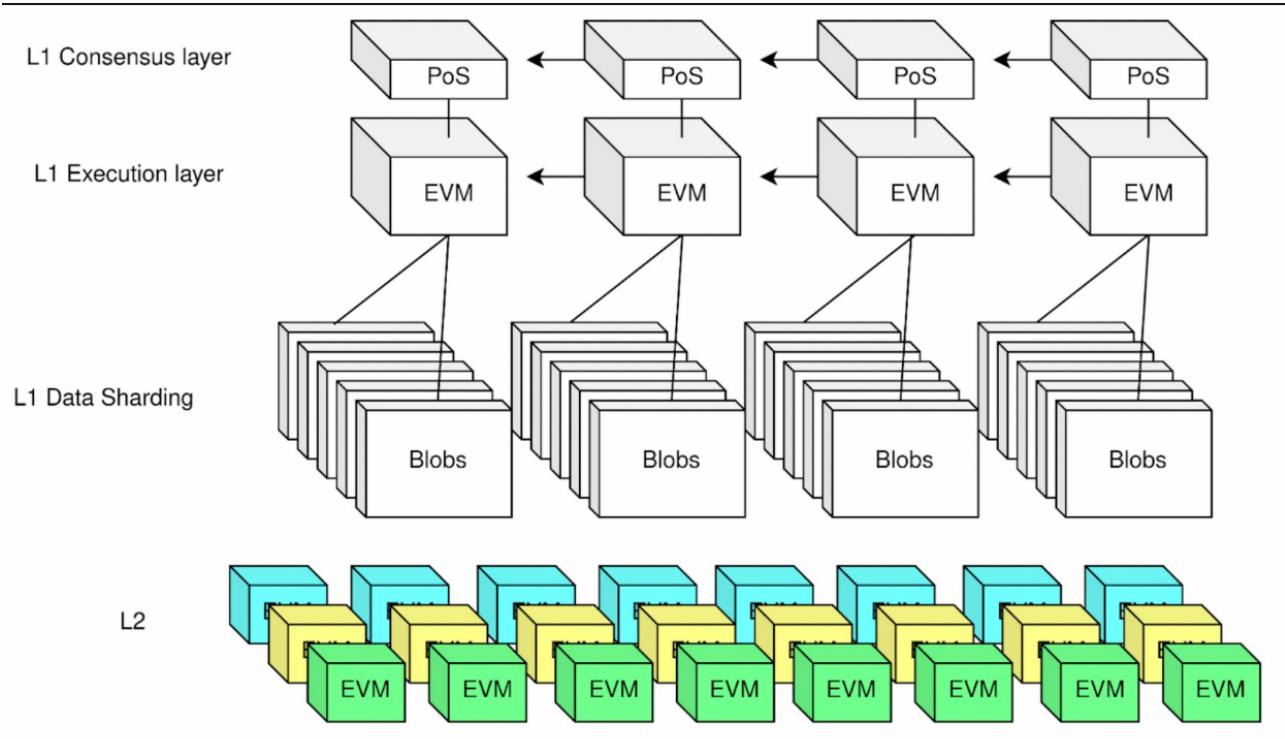


Figura 2.14: Esquema de la arquitectura de Ethereum que destaca la capa de sharding de datos basada en blobs, donde múltiples conjuntos de blobs almacenan datos de transacciones como soporte de capacidad para la ejecución en L1 y las soluciones de capa 2. Los blobs actúan como contenedores temporales de datos que alivian la carga de la EVM principal y permiten mayor escalabilidad. Fuente: [26]

2.6. Staking y validadores

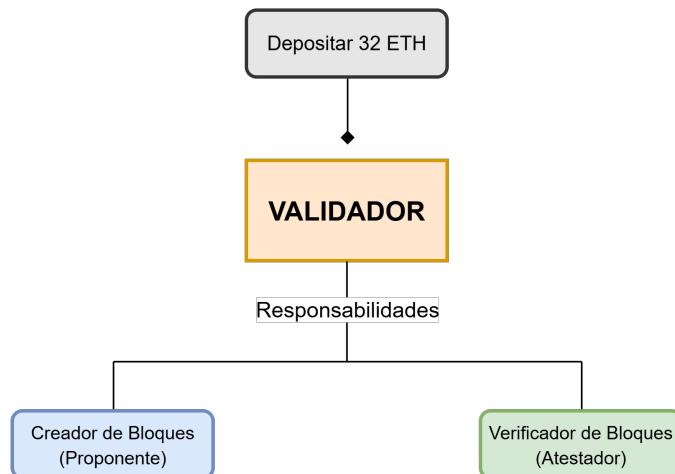
En Ethereum PoS, la seguridad de la cadena descansa en los validadores, participantes que apuestan (depositan) ETH para obtener el derecho y la responsabilidad de proponer bloques y votar sobre el estado de la cadena (fig. 2.15). Convertirse en validador requiere un depósito de 32 ETH en el Contrato de Depósito del protocolo (un smart contract especial en la red de ejecución) [12]. Una vez que alguien deposita 32 ETH allí, esos fondos quedan bloqueados en la Beacon Chain y el depositante espera su activación como validador. La activación no es inmediata; existe una cola de activación que limita la tasa de incorporación de nuevos validadores para preservar la estabilidad de la red [12]. Dependiendo de cuántos validadores haya actualmente activos, la espera puede ser de minutos u horas (en testnets) o incluso días (en caso de una afluencia masiva en mainnet). Cada epoch, el protocolo permite activar como máximo un número determinado de validadores nuevo (controlado por el parámetro activation_exit_churn_limit) [12]. Cuando le llega el turno, el validador pasa al estado activo y entra a formar parte del conjunto que participa en consenso.

Un validador activo debe ejecutar un software llamado a veces Cliente Validador (separado del Beacon Node pero vinculado a él) que firma mensajes cuando es necesario. Sus dos deberes principales son: proponer bloques cuando le toque (es seleccionado pseudoaleatoriamente como proponer en una ranura) y atestigar bloques ajenos (participar en los comités emitiendo votos llamados atestaciones) [2] [12]. La selección aleatoria de quién propone en cada slot y quién integra cada comité se realiza usando el mecanismo RANDAO (Random Beacon) mezclado con las apuestas de los validadores, garantizando imparcialidad y que todos los validadores tengan oportunidades proporcionales a su stake.

2.6.1. Atestaciones

Una atestación es básicamente el voto de un validador acerca de qué bloque fue el head de la cadena en su vista de la ranura asignada y acerca del checkpoint de epoch que considera válido. Es un mensaje que contiene: la raíz de bloque (hash) que el validador piensa que es la cabeza correcta de la cadena en ese momento, la referencia al checkpoint fuente justificado más reciente y al checkpoint destino (el de la epoch actual, que está votando para justificar) [13], junto con su firma y una indicación del comité al que pertenece [13]. Cuando un validador atestigua, está señalando “Yo creo que el bloque X es el correcto en esta ranura y apoyo que el checkpoint de epoch Y se finalice”. Estas atestaciones se propagan en la red y los clientes de consenso las agregan y las incluyen en bloques siguientes. Son la columna vertebral del consenso PoS: la participación alta de validadores (es decir, la mayoría enviando atestaciones correctamente cada epoch) es lo que permite justificar y finalizar la cadena de forma segura. En los paneles de monitoreo típicos, un indicador fundamental es el Porcentaje de Participación por epoch (qué fracción del total de validadores envió atestación válida en esa epoch). Valores cercanos al 99 % son normales en Ethereum mainnet saludable; si la participación cae por debajo, digamos,

CÓMO SER UN VALIDADOR



STAKING EN ETHEREUM

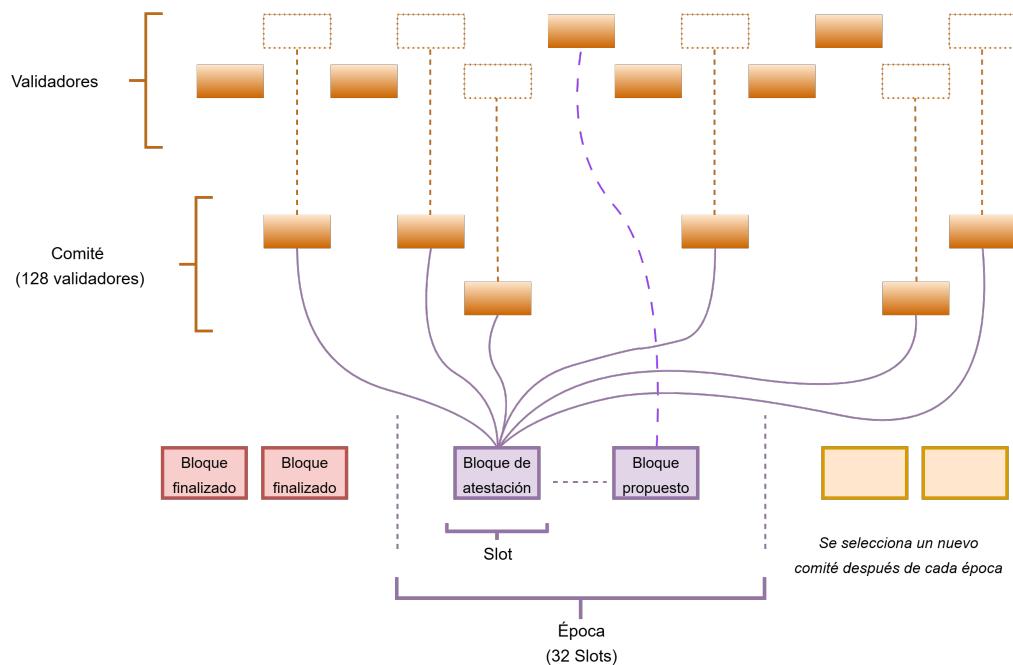


Figura 2.15: Infografía sobre el proceso de staking en Ethereum que ilustra cómo un participante se convierte en validador al depositar 32 ETH, asume los roles de proponente y atestador de bloques y participa en comités de 128 validadores que producen y finalizan bloques a lo largo de slots y epochs.

del 95 %, es señal de que muchos validadores están desconectados o teniendo problemas, lo que podría amenazar la finalización si cayera por debajo $\approx 67\%$. Por eso, más adelante al ver métricas veremos el rol crítico de las atestaciones: no solo marcan el head, sino que son garantía de liveness y seguridad.

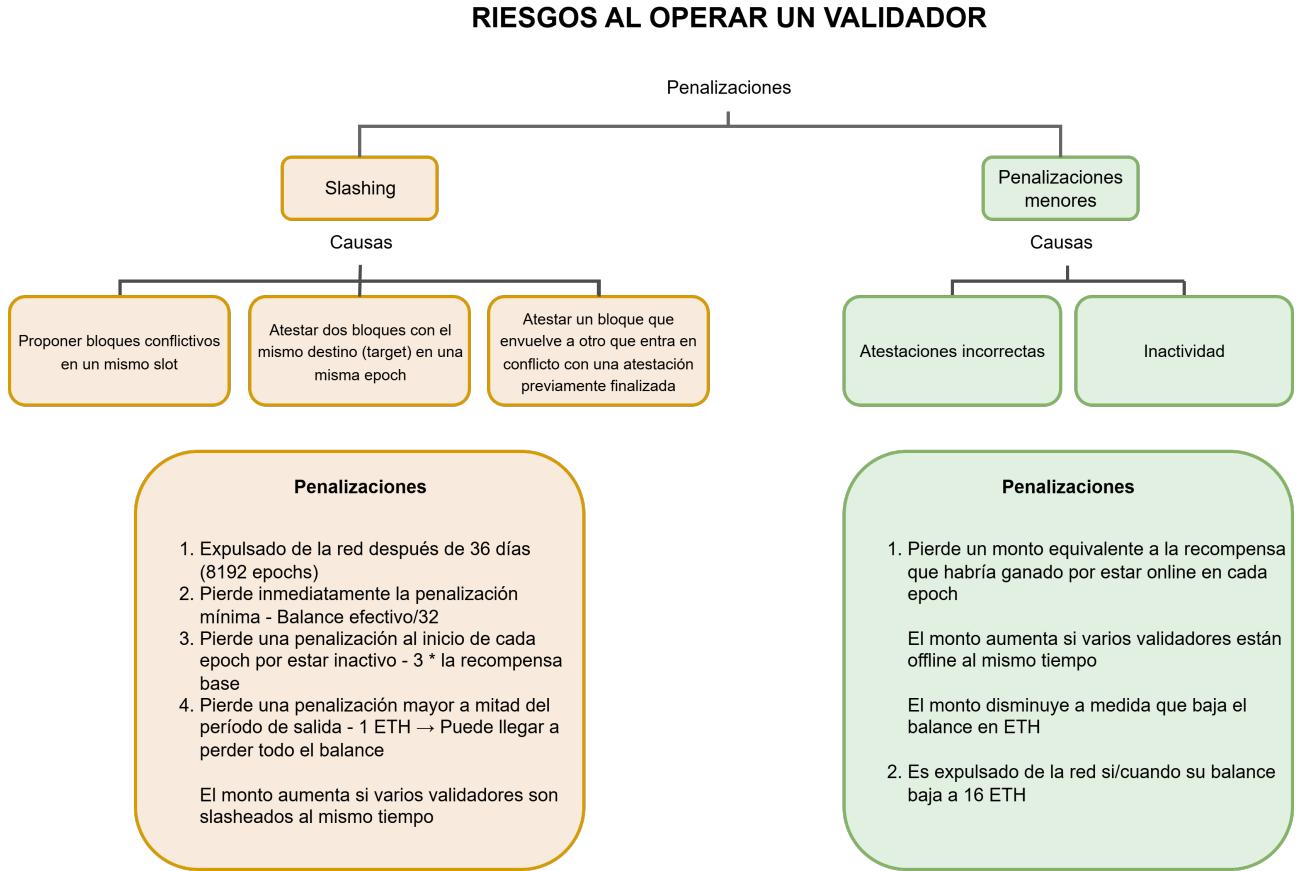


Figura 2.16: Diagrama de los riesgos al operar un validador en Ethereum que clasifica las sanciones en slashing y penalizaciones menores, detallando sus causas principales (bloques conflictivos, atestaciones incorrectas e inactividad) y las consecuencias económicas y de expulsión de la red asociadas.

Ahora bien, los validadores no están obligados a permanecer para siempre. Pueden retirarse voluntariamente a través de un proceso de exit. Un *voluntary exit* es un mensaje que el validador firma solicitando dejar de ser activo (esto también va en la Beacon Chain) [13]. Por protocolo, una vez que un validador anuncia su exit, debe esperar un tiempo mínimo (actualmente 256 epochs, ≈ 27 horas, llamado churn limit para salidas) para que se haga efectivo; después pasa al estado “exited”. Originalmente, antes de habilitar retiros, esos validadores salían pero su ETH quedaba bloqueado indefinidamente. Con la actualización Shanghai/Capella (2023) y la EIP-4895, se posibilitaron los retiros de stake: los validadores exited pueden recuperar su depósito y recompensas acumuladas. El mecanismo es push-based: la capa de consenso pone automáticamente en cola los retiros y los incluye en bloques de ejecución como operaciones especiales [28]. Hay dos tipos: retiros parciales (el validador continúa validando pero saca del protocolo el excedente sobre 32 ETH de sus recompensas, manteniendo solo 32 dentro) y retiros

totales (tras un exit, recupera los 32 ETH base más recompensas). Un validador debe tener actualizado su credential de retiro a un formato de dirección Ethereum (0x01) para poder recibir fondos en L1, a través de un mensaje BLSToExecutionChange. Estos detalles se mencionan porque en el monitoreo aparecen eventos asociados (eventos voluntary_exit cuando alguien sale, eventos bls_to_execution_change cuando alguien cambia su clave de retiro, y operaciones de retiro en los payloads cuando efectivamente se paga el ETH). Una métrica interesante es el flujo de entradas/salidas de validadores: un panel podría mostrar cuántos validadores nuevos entran vs. cuántos salen en un período, dando idea si el set activo crece o disminuye.

2.6.2. Recompensas y penalizaciones

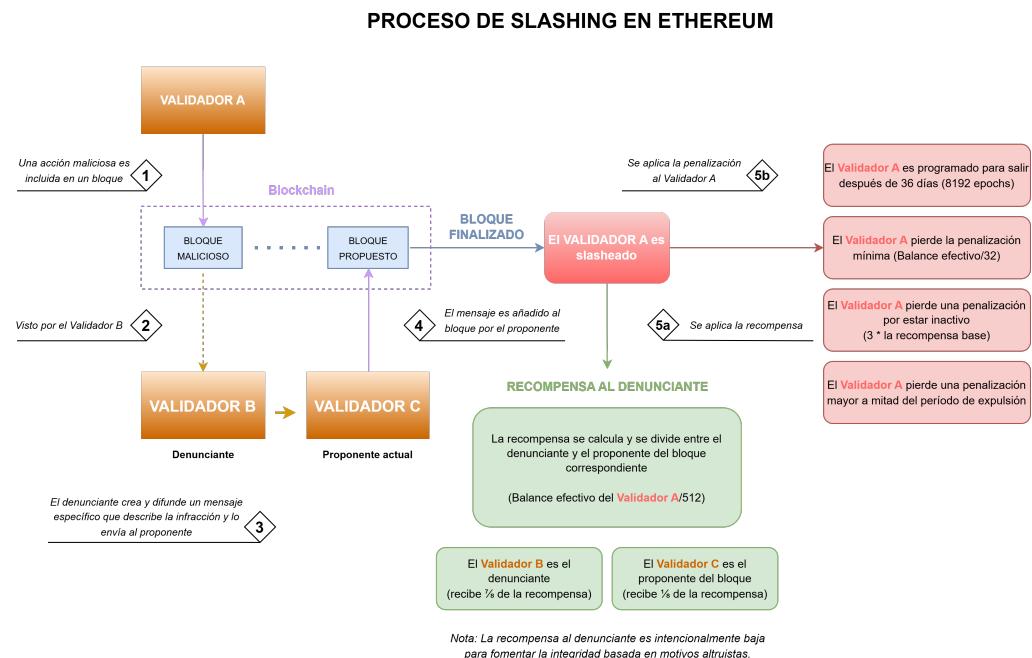


Figura 2.17: Diagrama del proceso de slashing en Ethereum que muestra cómo un validador comete una acción maliciosa, esta es detectada y reportada por un whistleblower, y el proponente incluye la evidencia en un bloque finalizado. Se detallan las penalizaciones aplicadas al validador sancionado y las recompensas otorgadas al whistleblower y al proponente del bloque.

Los validadores ganan recompensas en ETH por comportarse correctamente (proponer bloques válidos, atestaciones incluidas a tiempo, etc.) y pueden sufrir penalizaciones por inactividad o mal comportamiento (fig. 2.16) . La inactividad (no atestar) resulta en una pequeña pérdida de recompensa (básicamente un “no ganar” más que un castigo severo, excepto durante inactivity leak donde sí pierden capital gradualmente). Pero lo más notable son las penalizaciones por comportamiento deshonesto, conocidas como slashing. Hay dos condiciones principales que resultan en slashing a un validador: (1) Proposer equivocation – cuando un validador propone más de un bloque distinto para la misma ranura (solo uno puede ser honesto, el otro es una duplicación intencional o por error grave); y (2) Attester equivocation – cuando un validador emite dos atestaciones que se justifican mutuamente excluyentes (por ejemplo, afirmar

dos visiones diferentes de la cadena o de checkpoints en la misma epoch) [12]. Ambas acciones violan la regla de que un validador debe seguir una única historia de la cadena. El protocolo permite a cualquiera reportar estas faltas presentando evidencia (los pares de mensajes conflictivos), lo cual se incluye en un bloque como proposer_slashing o attester_slashing. Cuando un validador es “slashed”, es inmediatamente eyectado (pierde estatus activo) y su balance será recortado. La cantidad recortada inicia en un mínimo (por ejemplo 1 ETH) pero puede aumentar en función de cuántos validadores fueron slashed alrededor del mismo tiempo (penalización por correlación) [2]. En el peor caso, si muchos actuaron mal coordinadamente, el recorte para cada uno puede llegar hasta el 100% de su stake (todos sus 32 ETH) [2]. Esto disuade ataques organizados. Además, un validador slashed permanece en estado penalizado por \approx 36 días antes de ser expulsado definitivamente, perdiendo durante ese periodo una parte adicional diaria de su balance [2].

Para efectos de monitoreo, los eventos de slashing son sumamente importantes. En una red sana de Ethereum mainnet, los slashings son rarísimos (pueden pasar días sin uno, o solo alguno aislado causado por errores de configuración). La presencia de un proposer_slashing o attester_slashing en los eventos es una alerta roja: indica que algún validador cometió una falta grave. Un panel de seguridad podría mostrar el conteo de slashings por día; un pico inesperado (más de uno o dos) sería motivo de investigación inmediata. Por ejemplo, si muchos slashings ocurren juntos, podría implicar que un grupo grande de validadores estaba mal configurado (ej. usando la misma clave en dos sitios) o un intento coordinado de ataque fue reprimido – ambos escenarios críticos. Incluso un solo slashing usualmente amerita atención: normalmente es por error de operador, pero si se viera que varios validadores de una misma entidad están siendo slashed, se podría anticipar un problema mayor.

ESTADO ACTUAL DEL SLASHING

(Feb 2023)

Slasheados

- Actualmente, 223 validadores han sido slasheados desde que comenzó el staking el 1 de diciembre de 2020.
- Esto equivale a 2 validadores slasheados por semana.
- El primer slashing ocurrió después de solo 1 día y fue causado por una “infracción de reglas del proponente”.
- El mayor evento de slashing fue el 4 de febrero de 2021, cuando 75 validadores de la empresa Staked fueron slasheados.
- El evento multislashing más reciente fue el 23 de septiembre de 2022, donde 20 validadores fueron slasheados.
- Las infracciones más comunes giran en torno a las atestaciones y generalmente son resultado de errores de software o error humano.

Por ejemplo, para evitar tiempo de inactividad, algunos validadores sobrecomplican su infraestructura, como ejecutar nodos de respaldo simultáneamente. Esto puede provocar firmas dobles.



Figura 2.18: Infografía sobre el estado del slashing en Ethereum a febrero de 2023, que resume el número total de validadores sancionados, la frecuencia aproximada de slashing y los principales eventos históricos asociados, incluyendo grandes incidentes y cambios en las penalizaciones. Fuente: [29]

2.7. Eventos del *Beacon Node*

Una característica valiosa de los clientes de consenso Ethereum es que exponen un flujo de eventos de servidor (*Server-Sent Events*, SSE) a través del endpoint REST `/eth/v1/events` [19]. Este flujo permite suscribirse en tiempo real a las notificaciones que el nodo emite al ocurrir ciertos sucesos. Para monitoreo, es fundamental entender qué son estos eventos y qué indica cada tipo, ya que a menudo se configuran alertas o se alimentan paneles directamente con ellos. A continuación se describen los principales tipos de evento SSE disponibles, agrupados por su rol, enfatizando especialmente aquellos críticos como reorganizaciones y *slashings*.

2.7.1. Eventos de progreso de cadena

- **head:** Este evento se emite cuando el nodo ha elegido un nuevo bloque cabeza de la cadena (es decir, cuando la punta de la cadena cambia según el algoritmo de *fork choice*). Contiene datos como el *slot*, raíz del nuevo *head*, raíz del bloque anterior y si hubo transición de *epoch*, etc. En la práctica, cada vez que llega un nuevo bloque válido que se convierte en la mejor cabeza, se genera un *head*. Los paneles usan esto, por ejemplo, para saber en qué *slot/bloque* está actualmente la cadena seguida por el nodo [30]. Un *head event* también indica si esa cabeza está en modo *optimistic* (más adelante se explica el modo optimista) a través de un campo booleano. Para monitoreo rutinario, estos eventos son de alta frecuencia (aprox. uno cada 12s) y confirman que el nodo está recibiendo y adoptando nuevos bloques.
- **block:** El evento *block* se dispara cuando el nodo procesa cualquier bloque nuevo (ya sea que se convierta en *head* o no). Contiene típicamente el *slot* y la raíz del bloque recibido. En esencia, es una notificación de “llegó X bloque”. En una cadena saludable, la mayoría de bloques recibidos se vuelven el *head* inmediatamente, así que *block* y *head* suelen venir parejos; pero en caso de bifurcación puede haber eventos *block* para bloques que no terminan siendo la cabeza. Desde un punto de vista de monitoreo, *block* puede servir para contar bloques recibidos vs esperados (ver si hay *slots* perdidos) y diagnosticar bifurcaciones: por ejemplo, si en un mismo *slot* se reciben dos eventos *block* distintos, la red tuvo un *fork* en esa ranura.
- **finalized_checkpoint:** Este evento notifica cuando la cadena alcanza un nuevo *checkpoint* finalizado. Contiene la *epoch* y la raíz del bloque finalizado. Es enviado una vez por *epoch* (cuando finaliza, es decir, aproximadamente cada 6.4 minutos) en condiciones normales. Es extremadamente importante, ya que indica que hasta ese punto la historia es inmutable a menos que ocurra un ataque masivo. Un sistema de monitoreo registrará la llegada de cada nuevo *checkpoint* finalizado; si nota que pasa el tiempo y no llega ninguno nuevo (por ejemplo, han transcurrido más de dos *epochs* sin evento *finalized_checkpoint*), se encienden alarmas de falta de finalización. Así, este even-

to puede alimentar un panel mostrando la altura (*epoch*) finalizada más reciente y un temporizador de “tiempo desde finalización”. En *Ethereum mainnet*, debería haber un `finalized_checkpoint` regularmente; su ausencia, como se dijo, implica problemas serios.

- `chain_reorg`: Este evento señala una reorganización de la cadena en el nodo. Más concretamente, se emite cuando el nodo cambia su vista de la cabeza de la cadena no simplemente agregando un bloque nuevo en la punta, sino sustituyendo parte del final de su cadena por otra rama. Por ejemplo, si el nodo inicialmente tenía como *head* el bloque A en *slot* 100, pero luego recibe evidencia de que en el *slot* 99 había un bloque alternativo B con más apoyo, podría reorganizarse para adoptar B y su descendencia en lugar de A. El evento `chain_reorg` típicamente incluye información sobre a partir de qué *slot/raíz* ocurrió la divergencia. Este es un evento crítico: pequeñas *reorgs* (de profundidad 1 o 2 bloques) pueden suceder en la operación normal debido a la latencia de red, pero frecuentes *reorgs* o *reorgs* de mayor profundidad son señal de inestabilidad. Para el monitoreo, un aumento en la tasa de eventos `chain_reorg` puede indicar problemas de red (p. ej., un validador proponiendo sobre una base atrasada, o latencias que hacen que bloques tarden en llegar y se generen colisiones) [19]. Una reorganización involucra descartar uno o más bloques anteriormente aceptados en favor de otros, lo que potencialmente afecta métricas derivadas (como el número de transacciones confirmadas en un intervalo, etc.). Por eso, un sistema de monitoreo podría elevar un *warning* si detecta *reorgs* consecutivas o de profundidad inusual. De cara a análisis posteriores, entenderemos *reorgs* como “señales de inestabilidad o competencia en la propuesta de bloques”, lo cual puede correlacionar con bajadas en participación o con problemas en algún validador mayor.

2.7.2. Eventos de actividad de validadores

- `attestation`: Indica que el nodo recibió una atestación (no agregada) de parte de algún validador para cierta ranura. Cada atestación es un voto individual de un validador. Dado que hay miles de validadores, estos eventos son muy numerosos (en cada epoch, $32 * N_{\text{comités}}$ atestaciones potenciales). En la práctica, para monitoreo no se suelen procesar todos individualmente por su volumen, pero sí se podría usar para medir el retraso de atestaciones o cuántas se reciben tardías. Más frecuentemente, los dashboards se centran en atestaciones ya agregadas y contabilizadas, más que en este flujo crudo.
- `attestation_pool` (si existe, algunos clientes podrían emitir eventos de atestaciones agregadas al pool; en la especificación base se tiene el anterior).
- `contribution_and_proof`: Este evento está relacionado con las contribuciones de sincronización (sync committee contributions), introducidas en Altair para light clients. Un `contribution_and_proof` es análogo a una atestación pero emitida por los comités especiales de sincronización que cada epoch firman el estado para los light clients. Se emite

cuando el nodo recibe una contribución (o un agregado de ellas). En monitoreo, estos eventos reflejan la participación de los comités de sincronización, que asegura que los clientes ligeros pueden seguir la cadena de forma segura. Un descenso en estos también podría ser señal de problemas, aunque su importancia es menor para la mayoría de observadores que la de las atestaciones normales.

- **voluntary_exit:** Este evento indica que un validador ha iniciado una salida voluntaria. Contiene el identificador del validador y la epoch en la que será elegible para salir. Para la operatividad, esto señala cambios en el conjunto de validadores: cada *exit* reduce (eventualmente) el número de validadores activos. En un panel, podríamos mostrar el conteo de salidas en las últimas X horas. Individualmente, un *exit* no es alarmante (puede haber múltiples razones legítimas para salir: mantenimiento, reestructuración de fondos post-retiro, etc.), pero un número elevado de salidas simultáneas sí podría ser significativo (¿por qué muchos validadores quieren salir a la vez? – puede indicar incentivos económicos cambiantes o temor a algún problema). Tras Shanghai, hubo una avalancha de exits cuando se permitieron retiros [31], monitorear este evento permitió verificar que se procesaran según lo esperado. Un sistema de monitoreo también podría correlacionar eventos de **voluntary_exit** con *withdrawals* efectivos en la capa de ejecución posteriormente (una especie de tracking de que los salientes recibieron su ETH de vuelta).

2.7.3. Eventos de seguridad (críticos)

- **proposer_slashing:** Notifica que se ha incorporado una evidencia de slashing de un proponente. Esto significa que algún validador fue descubierto proponiendo dos bloques en la misma slot (equivocación de proponente). Como se explicó en la sección 2.6, este es un evento de seguridad grave: el validador será penalizado y expulsado. En términos de monitoreo, un evento de **proposer_slashing** es inmediato motivo de alerta. Deberíamos registrar el índice del validador slasheado, posiblemente identificar si pertenece a algún operador conocido (a veces pools de staking anuncian si tienen *slashings*), y evaluar si es un caso aislado o parte de algo mayor. Un panel podría tener un indicador “Slashing detected!” con detalles. Dado que estos eventos deben ser extremadamente raros, cualquier aparición es significativa.
- **attester_slashing:** Similar al anterior pero para slashings de attestadores: indica que se encontró un par de atestaciones conflictivas que implican a uno o más validadores (puede involucrar hasta 2 validadores en un attester slashing evidenciado). También es crítico por las mismas razones. A menudo, un mal comportamiento de atestación (enviar dos votos contradictorios) puede involucrar a más de un validador si compartían clave (lo cual no debería pasar) o si fue coordinado. En la historia de Ethereum PoS ha habido muy pocos slashings, por lo que su monitoreo es más binario (ocurrió / no ocurrió) que estadístico. Ambos tipos de slashing events en un dashboard estarían probablemente listados en una

tabla de incidentes de seguridad.

Tal como se analizó, los eventos que finalizan en “slashing” son señales de alerta máxima en monitoreo, ya que representan ataques o fallos graves del participante. Un sistema bien configurado enviaría notificaciones inmediatas al operador si detecta estos eventos en la alimentación SSE, dado que incluso un solo slashing puede dañar la reputación (y capital) de un pool de staking o indicar un intento de ataque siendo castigado.

2.7.4. Eventos de interfaz CL-EL y datos

- `payload_attributes`: Este evento es menos intuitivo a primera vista. Aparece en el contexto de la preparación de bloques cuando el cliente de consenso, actuando como un validador que va a proponer, envía al Execution Engine un mensaje Engine API llamado `engine_preparePayload` (o similar, antiguamente `engine_forkchoiceUpdated` con `payloadAttributes`). El evento `payload_attributes` en SSE puede reflejar que el validador local está solicitando construir un nuevo payload de ejecución para la siguiente ranura. En otras palabras, si nuestro nodo es un validador, unos segundos antes de su turno de propuesta, indicará los parámetros del payload (timestamp, randao, sugerencia de coinbase, etc.). No todos los clientes exponen esto claramente, pero de estar presente, marca el momento en que el nodo se prepara para forjar un bloque. En monitoreo, podría utilizarse para medir latencia entre la petición de bloque al *Execution Engine* y la recepción del bloque (aunque normalmente se confiaría en métricas internas para eso). Para una vista general, no es un evento crítico para la red en su conjunto, sino introspectivo de la operación del nodo validador local.
- `bls_to_execution_change`: Evento introducido con *Capella*, indica que un validador cambió su credencial de retiro de tipo `BLS` (0x00, usado antes para solamente acumular rewards dentro de Beacon) a tipo `Execution` (0x01, con dirección ETH para retirar). Estos cambios son prerequisito para poder retirar fondos; muchos validadores los enviaron tras Shanghai. Desde la perspectiva del monitoreo, estos eventos muestran la dinámica de preparación de retiros. En un panel podríamos ver cuántos validadores están cambiando sus credenciales por epoch. Un pico al alza puede ocurrir tras anuncios importantes (ej.: la apertura de retiros). Si bien no es “operativamente crítico” en el sentido de una falla, proporciona visibilidad sobre el comportamiento de los stakers. Por ejemplo, tras Capella se monitoreó cuántos de los validadores cambiaban su credencial [32]. Saber esto ayuda a entender cuántos validadores están listos para retirar en cualquier momento.
- `blob_sidecar`: Este es un nuevo tipo de evento previsto para cuando EIP-4844 esté activo. Dado que los blobs de datos viajan separados del bloque principal (como “sidecars”), el cliente de consenso emitirá `blob_sidecar` events al recibir cada blob asociado a un bloque. En la práctica, por cada bloque con blobs, existirán de 1 a N objetos blob sidecar que llegan vía la red P2P. El evento contendrá identificadores del blob (a qué slot y

bloque pertenece). Monitorear `blob_sidecar` es esencial para verificar la disponibilidad de datos para los blobs: si un nodo no recibe ciertos blob sidecars, significa que podría haber un problema de red o, peor, de disponibilidad (lo cual pondría en aprietos a `rollups` consumidores de esos datos). En un contexto de vigilancia, uno podría contar `blob_sidecars` recibidos vs esperados. También este evento daría input a un panel sobre “tráfico de datos de blobs”: por ejemplo, medir el tamaño total de blobs recibidos por minuto. En cuanto a criticidad, la falta de un `blob_sidecar` donde debería haberlo es grave (podría implicar que no todos los validadores obtuvieron los datos de un blob, lo que en el diseño normal haría que ese bloque no alcanzara finalización si el blob no se difundió correctamente). Por tanto, monitores en la era danksharding pondrán atención a estos eventos para garantizar que la capa de disponibilidad de datos está funcionando.

2.7.5. Eventos de clientes ligeros y gossip

- `light_client_finality_update` y `light_client_optimistic_update`: Son eventos especiales que proporcionan en un formato condensado la información de finalización y de cabeza de cadena para los clientes ligeros. Un `finality_update` para *light clients* contiene esencialmente lo necesario (firma agregada de sync committee y estado resumido) para que un light client sepa qué checkpoint fue finalizado. Un `optimistic_update` hace algo similar para la cabeza de la cadena (aunque no finalizada). Estos eventos no impactan la cadena en sí misma (no son “acciones”, sino notificaciones), pero su presencia indica que el nodo está generando y disponibilizando las pruebas para *light clients*. Monitorear si se están emitiendo puede ser útil para, por ejemplo, servicios que alimentan light clients (asegurarse de que el nodo produce esas actualizaciones). No suelen aparecer en paneles generales salvo que se esté evaluando la funcionalidad de light clients específicamente.
- `block_gossip`: Este evento (expuesto al menos por algunos clientes) indica la recepción de un bloque vía gossip de la red P2P. A diferencia de `block` que puede activarse cuando el bloque se procesa internamente, `block_gossip` denota puramente la capa de networking. Puede incluir timestamps o demoras. ¿Por qué es útil? Porque permite medir la latencia de propagación de bloques. Un nodo puede comparar el timestamp de cuando debería haber llegado un bloque (ej. inicio de slot + delta) con cuándo efectivamente su peer se lo entregó. Si se observan `block_gossip` muy retrasados, podría señalar problemas de red o una mala conectividad de peers. En monitoreo avanzado, se podría por ejemplo trazar un histograma de la distribución de llegadas de bloques. También, un `block_gossip` sin correspondiente head podría sugerir que el bloque fue huérfano (no canónico). En suma, este evento sirve para diagnóstico de gossip P2P, complementando métricas como número de pares y ancho de banda. No es tanto para alertas críticas sino para análisis de performance de la red.

De forma general, el flujo de eventos SSE del Beacon Node actúa como un “bus de notifica-

ciones” en tiempo real del comportamiento del nodo y de la red. Hemos destacado especialmente los eventos `chain_reorg` y `slashings` (`proposer_slashing` / `attester_slashing`) por su criticidad: los primeros detectan inestabilidad o forks, y los segundos ataques o errores graves de validadores. En un sistema de monitoreo, esos eventos probablemente estén conectados a alarmas con baja tolerancia (una sola ocurrencia puede disparar una advertencia o, al menos, un registro destacado). Mientras tanto, los eventos rutinarios como `head`, `block` y `finalized_checkpoint` constituyen el pulso normal de la cadena: su seguimiento permite verificar la *liveness* (por ejemplo, si no llegan `head events` en 30s, probablemente el nodo perdió conexión o está retrasado) y la consistencia (por ejemplo, comparar la raíz del `head` con la esperada en un explorador público). Eventos como atestaciones y *contributions* reflejan la participación, y los de cambios en validadores (`voluntary_exit`, `bls_to_execution_change`) reflejan la dinámica de *staking*. Finalmente, eventos de la interfaz CL-EL (`payload_attributes`, `blob_sidecar`) vinculan la operación interna entre clientes, importante para detectar problemas de sincronización entre capas.

Al comprender cada evento desde el punto de vista conceptual, resulta posible interpretar con mayor rigor los paneles de monitoreo. Un panel podría, por ejemplo, incluir una sección de “Eventos críticos recientes” que liste reorganizaciones y `slashings` junto con sus marcas de tiempo y detalles; otra sección de “Finalidad” que muestre el *epoch* finalizado actual (derivado de los eventos `finalized_checkpoint`); así como gráficas de “Participación vs. expectativa” calculadas a partir de atestaciones incluidas frente a las posibles. En la sección siguiente se analizará también cómo la salud del nodo (sincronización, número de *peers*) influye en la recepción de estos eventos: un nodo atrasado no emitirá `head events` actualizados, y uno sin *peers* apenas generará eventos hasta reconectarse. En conjunto, el conocimiento de los eventos SSE completa el panorama conceptual del monitoreo de Ethereum.

2.8. Salud del nodo

Monitorear Ethereum no se limita a observar la red en abstracto; es vital asegurarse de que el nodo desde el cual obtenemos los datos está sano y bien conectado. Un nodo podría estar desactualizado o aislado de la red y, en tal caso, sus métricas y eventos no reflejarían la realidad de la cadena en tiempo presente. Por ello, cualquier tablero de monitoreo debe incluir indicadores de la salud del nodo local. Entre los aspectos clave a vigilar están: el estado de sincronización, el modo optimista del cliente de consenso, la conexión con la capa de ejecución, la distancia en bloques/slots respecto a la cabeza de la red, y la conectividad P2P (*peers*). A continuación, detallamos cada uno y su importancia.

- `is_syncing`: Este es un indicador binario que refleja si el nodo de consenso está en proceso de sincronización o no [33]. Cuando un nodo arranca por primera vez o ha estado offline por un tiempo, debe descargar y verificar todos los bloques hasta alcanzar la punta de la cadena; durante ese proceso, `is_syncing` será `true`. Un nodo sincronizando no provee

datos de estado actuales de la red (está atrasado), por lo que la mayoría de métricas normales (tasa de bloques, participación, etc.) no tienen sentido hasta que termine. Monitorear `is_syncing` permite saber cuándo el nodo está listo. Asimismo, si en un nodo ya operativo `is_syncing` cambia a `true` repentinamente, es señal de un problema: podría indicar que el nodo tuvo que reiniciar o que perdió tantos datos que tuvo que re-sincronizar (lo cual es inusual). Normalmente se espera `is_syncing: false` en operación continua. En un panel se podría poner un semáforo “Nodo sincronizado / no sincronizado”.

- `head_slot` y `sync_distance`: Relacionado al anterior, los clientes de consenso proveen métricas de qué tan lejos están de la punta de la red. `head_slot` nos dice hasta qué slot ha llegado la cadena localmente [33]. `sync_distance` típicamente indica la diferencia entre el slot de la cabeza conocida de la red y el slot en que está nuestro nodo [33]. Por ejemplo, `sync_distance: 0` significa que nuestro nodo está en la punta actual; `sync_distance: 64` significaría que vamos 64 slots (unos ≈ 13 minutos) atrasados. Durante sincronización inicial, esta distancia empieza grande y va bajando a medida que se descargan bloques. En estado saludable *post-sync*, la distancia debería permanecer cero casi todo el tiempo (excepto breves momentos si hay un pequeño retraso en recibir algún bloque). Un panel puede mostrar “*Sync Distance (slots)*” en tiempo real. Si esa distancia empieza a crecer inesperadamente, implica que el nodo está cayendo detrás (un signo de alerta). Podría ser por problemas de peers (no recibe bloques a tiempo) o saturación del nodo. También es útil para trackear la sincronización inicial: un gráfico puede mostrar la distancia decreciendo hasta 0.
- `is_optimistic`: Este campo indica si el nodo de consenso está actualmente en un modo optimista respecto a la ejecución [33]. Modo *optimistic* ocurre cuando la capa de consenso ha adoptado un bloque como *head* para propósitos de seguir adelante, pero aún no tiene confirmación de la capa de ejecución de que el *Execution Payload* de ese bloque es válido. Esto puede suceder si el cliente de ejecución está atrasado, *offline*, o tardando en verificar un *payload* (por ejemplo, durante un *upgrade* o si hubo un fallo). En modo optimista, el nodo Beacon continúa participando en la red (no se queda congelado esperando), pero marca las cabezas como “provisionales” hasta que la EL valide. Si el *Execution Engine* luego rechaza un *payload* (porque era inválido), el nodo de consenso retrocedería ese bloque. Entonces, `is_optimistic: true` significa que hay cierto riesgo en los datos más recientes del *head*, ya que la ejecución no los respaldó aún. Monitorear esta bandera es fundamental para la confianza en los datos: si un nodo está en estado optimista, uno debe saber que los bloques *heads* que reporta podrían cambiar. Un panel puede resaltar “Modo optimista activado: Sí/No”. Lo ideal es que esto sea siempre No (`false`) durante la operación normal. Si se vuelve `true`, indica problemas en la conexión con la capa de ejecución (o en la ejecución misma). Podría ser que el *Execution Client* se haya caído – en cuyo caso el *Beacon Node* seguirá recibiendo bloques pero no puede ejecutarlos, entrando en modo optimista. La persistencia de este estado por más que unos pocos segundos es

grave: esencialmente la mitad de la funcionalidad del nodo (la ejecución) está ausente.

- **el_offline:** Muchos clientes proporcionan un indicador explícito de si la capa de ejecución se encuentra desconectada u *offline* [33]. Por ejemplo, Teku expone `el_offline` como un valor *true/false*. Este indicador complementa a `is_optimistic`: típicamente, si el *Execution Client* no responde, el *Beacon Node* marcará `el_offline: true` y, en consecuencia, `is_optimistic` también será `true`. Desde la perspectiva de monitoreo, si `el_offline` es `true`, ninguna métrica de ejecución (por ejemplo, estado de transacciones o *payloads*) será actual, y el nodo de consenso podría fallar al proponer bloques válidos (porque no podrá obtener un *payload*). Se trata de uno de los peores estados posibles para un nodo validador. Una alerta “*Execution Client desconectado*” debería dispararse casi de inmediato si esto ocurre. Las causas pueden incluir: proceso del cliente de ejecución caído, problemas de red entre CL y EL (si se ejecutan en máquinas separadas), incompatibilidad de versiones tras una *fork*, entre otras. Detectar este estado tempranamente es crucial para remediar la situación y evitar penalizaciones o fallas.
- Conectividad P2P (`peer_count`, `peers`): Ethereum es una red P2P; un nodo aislado no servirá de mucho. Por eso, monitorear la cantidad y calidad de peers es esencial. El endpoint `/eth/v1/node/peer_count` devuelve el número de pares separados por estado: cuántos conectados, cuántos en proceso de conexión, desconectados, etc.[34]. Por ejemplo, un nodo podría tener: 50 peers conectados, 5 connecting, 0 disconnecting, 0 disconnected (esto último usualmente cuenta peers conocidos pero actualmente no conectados). También está `/eth/v1/node/peers` que lista detalles de cada peer (ID, dirección, cliente, etc.), útil para diagnosticar diversidades de clientes o ubicaciones. En términos de monitoreo, `peer_count.connected` es la métrica principal: indica con cuántos otros nodos nuestro cliente está intercambiando información activamente. La capa de consenso de Ethereum por defecto intenta conectar un rango de 50 a 100 *peers*; La capa de ejecución suele también ser decenas. Si ese número cae muy bajo (digamos menos de 5), el nodo puede sufrir aislamiento: tardará en recibir nuevos bloques y atestaciones (lo que podría manifestarse en `sync_distance` aumentando, reorgs, etc.). Un valor de peer count de 0 significa que el nodo está completamente desconectado de la red p2p, básicamente no verá bloques nuevos hasta que consiga pares. Esto es crítico: un panel debe resaltar si `peer_count` baja de un umbral. Podría ser un problema de red local o algún ban masivo. Además, ver la variación en el tiempo: ¿los peers se mantienen estables o fluctúan? Caídas abruptas podrían indicar problemas (p. ej., nuestra IP fue baneada, o un error de configuración).
- Calidad de *peers*: no solo importa la cantidad, sino también su calidad. Aunque no exista un evento directo para esto, pueden derivarse métricas relevantes. Por ejemplo, la latencia promedio de recepción de bloques (que puede estimarse comparando el *timestamp* de *block gossip* con los tiempos de *slot*) está fuertemente vinculada a la calidad de los *peers* (peers más cercanos o con mejor conectividad implican menor latencia). Asimismo, la diversidad

de clientes entre los *peers*, que puede obtenerse listando los *peers* conectados, es relevante para la resiliencia: si todos los peers de consenso utilizan un mismo cliente y dicho cliente presenta un fallo, el nodo podría recibir información sesgada. Un sistema de monitoreo avanzado podría mostrar explícitamente la proporción de clientes presentes entre los *peers*.

- Otras métricas de salud: además de los indicadores específicos del protocolo, resulta relevante considerar métricas de sistema como el consumo de recursos (CPU, memoria) del nodo, ya que un nodo exhausto tiende a atrasarse. Aunque este aspecto pertenece al monitoreo de infraestructura, se intersecta directamente con la salud del cliente: un nodo con 100 % de uso de CPU, por ejemplo, podría comenzar a reducir su *peer count* o a demorar el procesamiento de bloques. Asimismo, en Ethereum los errores de configuración del reloj del sistema (marcas de tiempo) pueden ser problemáticos: un host con un reloj significativamente desviado respecto al tiempo real puede ser rechazado por otros nodos o generar bloques con *timestamps* inválidos. Por este motivo, algunos despliegues monitorean explícitamente la deriva del reloj frente a un servidor NTP de referencia.

En conjunto, la salud del nodo provee el contexto para interpretar todas las demás métricas de manera confiable. Si un tablero de Grafana indicara algo extraño en la participación o en las tarifas, primero debemos verificar: ¿Está nuestro nodo en buen estado para asegurar que no es un artefacto local? Por ejemplo, supongamos que un panel muestra repentinamente 0 bloques en 5 minutos. Antes de concluir “la red se detuvo”, hay que ver indicadores de salud: ¿Se nos cayeron los peers? ¿Nos quedamos desincronizados? ¿El cliente de ejecución está offline? Muchas veces, la explicación estará allí.

Por eso, es práctica recomendada dedicar una sección inicial del tablero a “Status del Nodo”: sincronizado o no, *slot* actual vs *slot* de red, *peers* conectados, uso de recursos. Solo si todo allí está en verde, podemos confiar en que los datos de rendimiento/red que vemos son genuinos. En la operación cotidiana, estos indicadores ayudan a darse cuenta de problemas tempranamente: por ejemplo, un operador podría ver que `peer_count` va disminuyendo lentamente, e investigar antes de que llegue a 0 y afecte más cosas.

En las siguientes interpretaciones de paneles, supondremos siempre que nuestro nodo está sano o notaremos inmediatamente si un problema de nodo es la causa de una métrica. Así, estos conceptos sobre sincronización, modo optimista, capa de ejecución *offline*, distancia de sincronía y *peers* nos permitirán evaluar la fidelidad de los datos de monitoreo y también entender qué medidas tomar: por ejemplo, si `el_offline` es `true`, sabremos que ningún nuevo bloque se está ejecutando en nuestro nodo, y la prioridad será restaurar la conexión de la EL, en lugar de buscar fallos en la red global.

2.9. Parámetros de red relevantes

Ethereum existe en múltiples redes o cadenas paralelas: la principal es *Mainnet* (la red principal de producción), pero hay redes de prueba como *Goerli*, *Sepolia* u otras (y también

bifurcaciones privadas, redes de desarrollo, etc.). Cada red se define por un conjunto de parámetros de cadena: un identificador de red o chain ID, un génesis (bloque inicial) y una secuencia de bifurcaciones (forks) programadas que determinan su protocolo actual. Es crucial al monitorear estar consciente de qué red estamos monitoreando y cuáles son sus parámetros específicos, porque interpretar los datos correctamente depende de ello.

- Identidad de red (*chain ID* y nombre de red): la red principal de Ethereum (*Mainnet*) tiene `chainId = 1` y suele identificarse simplemente como “Mainnet”. Las redes de prueba poseen otros identificadores (Goerli utilizaba 5, Sepolia 11155111, entre otros). Un nodo de Ethereum conoce internamente su génesis y su *chain ID*, y normalmente la API JSON-RPC expone esta información mediante el método `eth_chainId`. Monitorear este parámetro equivale a comprobar que el nodo está conectado a la red esperada. Aunque pueda parecer un detalle menor, se han dado casos en los que un tablero de monitoreo estaba apuntando a una testnet mientras se asumía que era Mainnet. En paneles multi-entorno resulta útil mostrar explícitamente la red, por ejemplo “Red: Mainnet (`chainId 1`)”. Dado que el *chain ID* solo cambia si se reconfigura el nodo, se trata de un parámetro esencialmente estático.
- *Forks* y *upgrades* activos: Ethereum se actualiza mediante *hard forks* periódicos, cada uno definido por un conjunto de EIP. Ejemplos recientes son Bellatrix (preparó *The Merge*), Paris (ejecutó *The Merge*), Shanghai/Capella (habilitó retiros) y, en el futuro, Deneb/Cancún (EIP-4844). Cada *fork* se activa en un bloque o *epoch* específico. Los nodos deben actualizarse a versiones de cliente compatibles antes de cada *fork*; de lo contrario, permanecerán en una cadena incompatible. Desde la perspectiva del monitoreo, conocer qué *forks* están activos ayuda a interpretar la presencia o ausencia de determinadas métricas. Por ejemplo, antes de Capella no existían eventos de `withdrawal` ni `bls_to_execution_change`; antes de Bellatrix no había `execution_payload` en los bloques de consenso, ya que la Beacon Chain *pre-Merge* no incluía transacciones. En un panel histórico que abarque períodos previos a ciertos *forks* se observarán discontinuidades: por ejemplo, la “tarifa base” aparece recién a partir de EIP-1559 (agosto de 2021). Parámetros como tiempo de ranura de 12 s y *epoch* de 32 ranuras se han mantenido constantes en *Mainnet* desde el génesis de PoS, aunque en principio podrían diferir en redes especiales o testnets experimentales. Lo esencial es calibrar expectativas y umbrales de monitoreo según la red y la era de protocolo: en la *Mainnet* actual se esperan aproximadamente 7200 ranuras por día. Si una testnet utilizara otro tiempo de ranura, los cálculos deberían ajustarse.
- Mainnet frente a testnets: implicaciones en las métricas: las redes de prueba suelen contar con menos validadores que Mainnet (Goerli tenía aproximadamente 30.000, Sepolia 50.000, frente a más de 600.000 en Mainnet). Esto impacta directamente en métricas como la participación: una docena de validadores inactivos en *Mainnet* es prácticamente imperceptible, mientras que en una testnet pequeña puede representar alrededor de

un 1 % menos de participación. Además, las testnets pueden atravesar períodos sin finalidad por falta de operadores activos (por ejemplo, si numerosas entidades apagan sus nodos de Goerli simultáneamente, la red puede quedar temporalmente sin finalización). En Mainnet, un episodio de este tipo sería extremadamente grave; en testnet es relativamente frecuente. En consecuencia, los umbrales de alerta deben diferir: un panel sobre una testnet podría considerar aceptable una participación del 80 % sin generar alarmas, mientras que en Mainnet un valor así sería motivo de preocupación. De forma análoga, el valor de la *base fee* tiene significados diferentes: en testnet el gas no posee valor monetario, por lo que puede observarse una volatilidad poco representativa (usuarios realizando pruebas, llenando bloques sin coste real). En Mainnet, en cambio, la *base fee* tiende a reflejar demanda económica genuina. La interpretación de las métricas debe, por tanto, estar siempre contextualizada por la red de origen.

- Origen de los datos - nodo propio frente a proveedor: otra decisión relevante consiste en determinar si las métricas provienen de un nodo local operado por la propia organización o de un servicio de terceros (por ejemplo, Infura [35], Alchemy [36] u otros). En el segundo caso, ciertos detalles sobre la salud del nodo subyacente pueden no estar disponibles o ser abstraídos por el proveedor. Al mismo tiempo, aparecen nuevas fuentes de incertidumbre, como límites de tasa o colas internas que introducen latencia. Por ello, el panel debería señalar de forma explícita la fuente de datos, por ejemplo “Datos obtenidos desde Infura – Mainnet”. Ante anomalías, esta información ayuda a discernir si el origen probable es la red en sí o el proveedor. En este trabajo se asume, de forma general, la existencia de un nodo propio, pero es importante remarcar que la confianza en las métricas está condicionada por la confianza en la fuente de datos utilizada.
- Configuraciones específicas de cada red: la Mainnet de Ethereum presenta actualmente ciertos parámetros fijos: tiempo de ranura de 12 s, *epoch* de 32 ranuras, contrato de depósitos en la dirección 0x424..., entre otros. Otras redes utilizan contratos de depósito distintos (aspecto que no suele afectar a las métricas operativas), un estado génesis diferente y, a menudo, un volumen total de *stake* mucho menor. Por ejemplo, el balance total en *staking* en Mainnet es de aproximadamente 20 millones de ETH; en Sepolia es trivial y en Goerli fue puramente ficticio (en el orden de 20k GoETH). Conocer estas diferencias permite contextualizar las métricas: las recompensas por validador, expresadas en términos porcentuales, tienden a ser mayores en redes con menor *stake* total, como muchas testnets.

En la práctica, al diseñar un tablero de monitoreo suele incluirse un recuadro de “Información de red” con datos como el nombre de la red, el *chain ID*, la versión de *fork* actualmente activa (por ejemplo, “Shanghai/Capella active”) y, eventualmente, el número total de validadores (que puede obtenerse mediante la API, por ejemplo con `/eth/v1/beacon/states/head-/validators` y el parámetro `state=active`). Estos parámetros rara vez generan alertas, dado

que son estáticos o de cambio lento, pero ayudan a evitar interpretaciones erróneas. Por ejemplo, un saldo pendiente de retiro de 34 000 ETH tiene implicaciones muy diferentes si corresponde a Mainnet o a una testnet.

Asimismo, en la interpretación de paneles históricos, este contexto permite explicar la ausencia de determinadas métricas. Un panel puede incluir, por ejemplo, una sección “Blob usage”; en la Mainnet previa a Deneb esa sección aparecerá vacía o en cero porque EIP-4844 aún no está activo. Sin recordar el estado de los *forks*, podría confundirse “0 blobs” con “falta de datos”, cuando en realidad la funcionalidad todavía no existe en esa red.

Finalmente, resulta importante considerar las versiones de software de los clientes. Un nodo desactualizado (que ejecute una versión antigua) puede fallar en un *fork* y desviarse hacia una cadena incompatible. En la práctica, esto suele manifestarse como un nodo que deja de avanzar mientras el resto de la red continúa produciendo bloques. Por este motivo, entre los parámetros de red a vigilar se incluye la versión del cliente y, cuando está disponible, el `fork_digest` de la configuración actual, que puede correlacionarse con la especificación del protocolo. En entornos de producción es habitual monitorear que la versión del cliente se corresponda con la recomendada para los *forks* programados.

En síntesis, el mensaje central de esta sección es que el contexto de red resulta determinante para interpretar las métricas de monitoreo. A lo largo del resto del trabajo se asumirá, salvo indicación explícita, que las métricas se refieren a Ethereum Mainnet; sin embargo, las consideraciones expuestas aplican a cualquier red compatible con el protocolo de Ethereum. Identificar primero qué cadena se está observando y cuáles son sus particularidades (cantidad de *stake*, *forks* activos, parámetros de tiempo, etc.) permite formular juicios correctos sobre los datos y alinear las expectativas respecto del comportamiento esperado de la red.

2.10. Síntesis operativa

Habiendo desglosado todos los componentes principales –tiempo de la beacon chain, arquitectura de consenso/ejecución, modelo de gas, dinámica de validadores, eventos, estado del nodo– conviene integrarlos en un mapa mental operativo de cómo funciona Ethereum post-Merge día a día y cómo leer sus señales en conjunto. Imaginemos el ciclo continuo:

Cada 12 segundos transcurre una ranura en la que idealmente un validador propondrá un bloque. Ese bloque incluye un execution payload con transacciones, consumiendo cierto gas y pagando una base fee (que se ajustará luego según ese consumo). Los demás validadores en su turno atestarán ese bloque, contribuyendo a definir la cadena head actual. Pasan 32 ranuras y completa una epoch; los atestados acumulados permiten (asumiendo suficiente participación) marcar un checkpoint justificado y posiblemente finalizar la epoch previa. Mientras tanto, algunos validadores podrían estar entrando (tras su depósito) o saliendo (voluntary exit) del conjunto, lo que gradualmente cambia la cantidad total de stake activo. Durante este proceso, la Capa de Ejecución en cada bloque maneja las transacciones de usuarios: calcula el estado actualizado, quema las tarifas base y asigna propinas, produce logs de eventos de contratos,

etc. Si hay mucha demanda de transacciones, los bloques van llenos (`gas_used` cercano al límite) y la base fee sube; si la demanda baja, base fee baja y bloques quizá queden semivacíos. En paralelo, (próximamente) si rollups publican datos masivamente, los blobs llenarán su cuota y la base fee de blob gas subirá análogamente.

Todos estos procesos generan señales medibles: el Beacon Node emite eventos SSE cada vez que hay un nuevo head, cuando se finaliza un checkpoint, si ocurre un reorg, cuando recibe atestaciones, etc. Además, exponemos métricas numéricas: participación (% de validadores que atestaron cada epoch), duración de forks, tarifas promedio por bloque, número de peers, y más. Un panel de monitoreo cruza estas piezas para brindar una visión completa.

Al analizar datos de un sistema de monitoreo, hay ciertos patrones sanos que esperamos ver en una Ethereum funcionando correctamente:

- Finalización regular: Debería haber un nuevo bloque finalizado aproximadamente cada 6.4 minutos (2 epochs). En un dashboard, veríamos la altura de finalización incrementarse sin grandes lagunas temporales. Finality regular indica que más de 2/3 de validadores son honestos y están en línea.
- Alta participación de validadores: Idealmente encima del 98-99 %. En gráficos por epoch, la barra de participación estaría casi llena siempre. Esto muestra que casi todos los validadores están enviando sus atestaciones y la red está bien soportada.
- Casi cero reorgs y blocks huérfanos: A excepción de alguna rara reorganización de 1 bloque esporádica, la cadena head avanza linealmente. Los eventos `chain_reorg` serían muy infrecuentes (quizá algunos al día como mucho, o ninguno). Esto refleja baja latencia y sincronía entre validadores.
- Sin eventos de slashing: En condiciones normales, no debería verse ni `proposer_slashing` ni `attester_slashing`. Su ausencia es el estado deseado. Un historial limpio de slashings sugiere que todas las entidades están operando correctamente.
- Uso moderado de recursos: Los bloques suelen estar cerca del `gas target` pero no permanentemente en el `gas limit` máximo (a menos que la red esté continuamente al máximo de capacidad). La base fee puede fluctuar pero tiende a estabilizarse. Los `blob usage` (tras 4844) también se mantendrían cerca del target en promedio si hay demanda constante de rollups, pero raramente saturados al 100 % sostenidamente. Un patrón “diente de sierra” suave en base fee es normal; picos abruptos y mantenidos significan congestión sostenida.
- Nodo con buena salud: `peer_count` estable (por ejemplo, en torno a 50 `peers`), `is_syncing = false`, `is_optimistic = false`, `el_offline = false`. El nodo reporta `head_slots` cercanos al tiempo real (`sync_distance ≈ 0`). Esto indica que el punto de observación es confiable.

Si un monitoreo muestra todos esos aspectos en verde, la conclusión es que Ethereum está operando óptimamente y los usuarios deben ver confirmaciones rápidas, finalización garantizada en minutos, y ninguna anomalía de seguridad.

Por otro lado, debemos estar atentos a ciertas señales de riesgo o anómalas:

- No-finalización prolongada: Si pasan más de ≈ 15 minutos sin finalización, es un síntoma severo. En panel, veríamos que el último finalized checkpoint quedó estancado varias epochs atrás y posiblemente un contador “time since finality” volviéndose rojo. Esto puede indicar que más de un tercio de validadores están fuera de línea o una partición de red. Es prioritario en ese caso investigar, teniendo en cuenta, que ya ha pasado en mainnet un par de veces en mayo 2023 por problemas de clientes, se resolvió en minutos, pero es muy notorio en monitoreo [6].
- Aumento de chain_reorg eventos: Si el panel registra reorganizaciones frecuentes (ej. varias por hora o reorgs de profundidad mayores a 1), la estabilidad de la cadena está comprometida. Quizás la red sufre latencia alta (p. ej., un problema de gossip en cierto cliente). Si combinamos esto con latencias de bloque elevadas, confirmaría el problema. Los validadores pueden estar produciendo bloques conflictivos. Sería una situación a escalar a los desarrolladores de clientes.
- Caídas en atestaciones o en *sync committees*: si la participación disminuye bruscamente, por ejemplo de 99 % a 80 %, y se mantiene en ese nivel, ello indica que un conjunto significativo de validadores se encuentra desconectado o experimentando fallos (posiblemente por problemas en un proveedor de *staking*). Las consecuencias son una menor resiliencia del protocolo y un riesgo aumentado de que la red entre en períodos sin finalización si la participación desciende aún más. En los paneles, esto se visualizaría como barras de participación incompletas por epoch y debería estar vinculado a alertas tempranas, dado que la ausencia de finalización (epochs donde ningún *checkpoint* llega a ser finalizado) incrementa el riesgo de reorganizaciones profundas y, por ende, el riesgo para la seguridad de la cadena.
- Aparición de eventos de slashing: Cualquier slashing es relevante. Un panel lo destacaría “Slashing: Validator 12345 slashed at epoch X”. Un caso aislado indica una entidad tuvo un error (no amenaza a la cadena en su conjunto, pero sí a la entidad). Múltiples slashings simultáneos (ej. decenas) sería alarmante: sugiere un fallo masivo (por ejemplo, todos esos validadores corrían en duplicado). En extremo, un ataque coordinado detectado. Habría que ver si son todos de un mismo operador. Un monitor podría agrupar slashings por validator index o por firmas para estimar si son correlacionados.
- Problemas persistentes CL–EL: Si el nodo entra en `is_optimistic=true` durante muchos slots o con frecuencia, hay desconexiones repetidas a la EL. En panel, quizás veríamos saltos a modo optimista y luego recovery. Esto a nivel de red significa nuestro nodo no

está validando payloads a tiempo, si es generalizado (no solo nuestro nodo), implicaría problemas de clientes de ejecución con cierto bloque (peor escenario, cadena podría estancarse). `el_offline=true` nos diría puntualmente que la conexión se perdió. Este indicador prácticamente avisa “no confíes en este nodo hasta arreglarlo”.

- Alta distancia de sincronización: Si `sync_distance` comienza a crecer en un nodo antes sincronizado, su feed de datos empieza a quedarse atrás. Quizá se desconectó de peers (lo veríamos junto a `peer_count` bajo) o está sobrecargado. Indirectamente se detecta porque los eventos head dejarían de coincidir con el tiempo actual (slot actual en la red vs slot del nodo difiere).
- Peer count muy bajo: Un nodo con pocos peers puede volverse un “punto ciego”. Si en panel notamos `peer_count` disminuyendo (por debajo de 10, p.ej.), anticipamos problemas: tardará en recibir bloques, posiblemente incremente reorgs locales. Podría requerir reiniciar el nodo o investigar conectividad (firewalls, etc.).
- Métricas de recursos fuera de rango: Si estuviéramos monitoreando CPU/Mem/disco, su saturación podría preludiar fallas (no directamente de protocolo, pero sí del nodo). Por ejemplo, si el disco se llena, el nodo podría colapsar y caer en syncing nuevamente.

De manera integrada, lo relevante es establecer relaciones entre los distintos indicadores: un problema operativo suele manifestarse simultáneamente en varias métricas. Por ejemplo, una finalización retrasada combinada con una caída sostenida de la participación al 60 % apunta a que una fracción significativa de validadores se encuentra fuera de línea; en ese caso, la reducción de participación explica directamente la pérdida temporal de finalidad. Si, adicionalmente, se conoce por fuentes externas que un determinado cliente de consenso presentó un fallo, es razonable inferir que los validadores que lo ejecutan abandonaron la red de forma simultánea, provocando la degradación observada. De forma análoga, una *base fee* persistentemente elevada junto con bloques cercanos al *gas limit* indica un periodo de congestión sostenida de la red, típicamente asociado a eventos de alta demanda (por ejemplo, lanzamientos de tokens o acuñaciones masivas de NFT).

En este sentido, el marco conceptual desarrollado permite interpretar un panel de monitoreo de Ethereum como una representación coherente del estado de la red. En condiciones normales, la lectura conjunta de los indicadores debería ser consistente con un escenario en el que los bloques se producen con regularidad, las *epochs* se finalizan sin demoras significativas, la gran mayoría de los validadores participa de forma estable, no se registran incidentes de seguridad relevantes y el nodo local mantiene una conectividad adecuada, todo ello dentro de parámetros esperados de utilización de gas. Ante desviaciones de este patrón, las combinaciones de señales descritas en esta sección orientan el diagnóstico hacia causas probables, como congestión, problemas de clientes o fallos de infraestructura.

Este capítulo de marco teórico servirá, en secciones posteriores dedicadas al análisis de paneles concretos, para que cada gráfico o alerta pueda situarse en su contexto adecuado.

Se aplicarán directamente conceptos como “ranuras/*epochs*” a los ejes de tiempo de bloques, “*head* vs. *finalized*” a la latencia de confirmación, “*slashings*” a los indicadores de seguridad y “estado del nodo” a la confiabilidad de los datos observados. En suma, se ha construido la base conceptual mínima y suficiente para que un lector sin experiencia previa en Ethereum pueda interpretar indicadores de monitoreo post-*Merge* con criterio técnico y con el contexto necesario para emitir juicios fundados sobre el comportamiento de la red.

2.11. Trabajos relacionados y soluciones de monitoreo

En el ecosistema Ethereum existen varias aproximaciones de monitoreo centradas en telemedición o tableros públicos. La tabla 2.1 resume las más cercanas al alcance de este trabajo y señala brechas que la propuesta cubre (flujo SSE completo, ingesta ETL y paneles reproducibles).

Cuadro 2.1: Comparativa resumida de soluciones de monitoreo para Ethereum.

Solución	Cobertura de eventos SSE	Ingesta/ETL	Limitaciones frente a este trabajo
Exporters Prometheus de clientes (Teku, Lighthouse, Prysm)	No exponen <code>/eth/v1/events</code> ; sólo métricas internas del cliente (CPU, peers, sincronización).	Scraping Prometheus hacia TSDB; no procesan JSON de la Beacon API.	Observan salud de cliente e infraestructura, no eventos de consenso ni blobs/reorgs detallados.
Dashboards públicos (beaconcha.in, beaconstate.info)	Muestran <code>head</code> , finality, slashings y otras métricas agregadas; el stream SSE crudo no es accesible.	Pipelines de ingestión y almacenamiento propietarios; el usuario sólo consume las vistas ya construidas.	Servicios de sólo lectura y difícil autohospedaje; no permiten controlar retención ni agregar nuevas fuentes o reglas propias de alerta.
Soluciones comerciales (Infura/Alchemy observability)	Pueden ofrecer streams HTTP/WebSocket filtrados; la cobertura de tópicos y campos depende del proveedor.	Ingesta y paneles cerrados, con poca visibilidad sobre la transformación interna y el almacenamiento.	Dependencia de SaaS y modelo de costo por volumen; integración limitada con pipelines propios (por ejemplo, NiFi → InfluxDB) y baja reproducibilidad académica.
Web3j-eth2 (2021)	Cliente Java que implementa SSE para tópicos básicos (<code>head</code> , <code>block</code> , <code>attestation</code> , <code>finalized_checkpoint</code> , <code>chain_reorg</code>) de versiones tempranas de la Beacon API.	Consumidor SSE embedible en aplicaciones Java; no aporta ETL, almacenamiento ni dashboards listos para usar.	Proyecto poco mantenido y alineado con una versión antigua del protocolo; no cubre eventos recientes (por ejemplo, blobs) y exige construir todo el entorno de monitoreo alrededor.
Propuesta de este trabajo	Cubre explícitamente los tópicos SSE más relevantes (<code>head</code> , <code>finalized_checkpoint</code> , <code>chain_reorg</code> , <code>blob_sidecar</code> , entre otros), con posibilidad de extensión.	Librería Java para SSE más Controller Service y Processor de NiFi que apllanan los mensajes y los envían en Line Protocol a InfluxDB; dashboards de Grafana versionados y desplegados con Docker Compose.	Stack autohospedable, abierto y reproducible, orientado a experimentos y a extender métricas y alertas sin depender de servicios externos ni pipelines cerrados.

El aporte diferencial reside en la combinación de una librería abierta para SSE, un bundle de NiFi específico para eventos de consenso y dashboards versionados que pueden redeployarse sin dependencias propietarias, cubriendo eventos críticos (reorgs, slashings, blobs) que las alternativas anteriores omiten o tratan de forma agregada.

CAPÍTULO 3

Sistema bajo estudio

3.1. Visión general del proyecto

El sistema se compone de un conjunto de servicios que trabajan de forma coordinada en un contenedor de Docker, con una red interna denominada monitoring_net. Cada servicio cumple una función específica dentro del flujo de monitoreo

- Apache NiFi 2.4.0 [37]: encargado de orquestar la ingestión, transformación y enruteamiento de los datos hacia la base de datos.
- Librería Java 21 Beacon Chain SSE [8]: una librería dedicada a consumir eventos de la Beacon API de Ethereum a través de un nodo. Fue desarrollada como parte del presente trabajo y tiene como objetivo permitir que el procesador en Java mantenga activa la conexión al endpoint SSE /eth/v1/events, gestionando la suscripción, la reconexión automática y el manejo continuo de los flujos de eventos. Esta librería se integra directamente con un procesador y controlador personalizado de NiFi, el cual también fue desarrollado como parte del trabajo [9], facilitando el procesamiento en tiempo real de la información proveniente de la red Ethereum.
- InfluxDB2.7.11 [38]: base de datos de series temporales utilizada para almacenar los eventos procesados y las métricas del sistema.
- Grafana 12.0.0 [39]: herramienta de visualización que permite crear paneles interactivos (dashboards) y configurar alertas basadas en las métricas almacenadas. Se encuentra configurada para habilitar el acceso anónimo con permisos de solo lectura, lo que facilita la exposición pública de las visualizaciones.

- Nginx (alpine) [40]: actúa como proxy reverso para Grafana. El contenedor utiliza certificados almacenados localmente y un archivo de configuración personalizado, permitiendo la futura publicación segura del entorno en Internet.

3.2. Arquitectura general del sistema

La arquitectura del entorno sigue un modelo de flujo de datos orientado a la observabilidad:

1. Los datos son recolectados y procesados dentro de Apache NiFi, donde se aplican transformaciones, normalizaciones y reenvíos hacia el motor de almacenamiento.
2. La librería Java 21 Beacon Chain SSE mantiene la conexión activa con el nodo Ethereum y provee el flujo continuo de eventos al procesador Java integrado en NiFi.
3. InfluxDB actúa como repositorio central de métricas y eventos, utilizando estructuras optimizadas para consultas de series temporales.
4. Grafana consume los datos almacenados en InfluxDB y los presenta al usuario mediante paneles dinámicos, permitiendo además de definir umbrales de alerta.
5. Finalmente, Nginx expone el servicio de Grafana al exterior bajo HTTPS, garantizando una interfaz segura para la visualización pública.

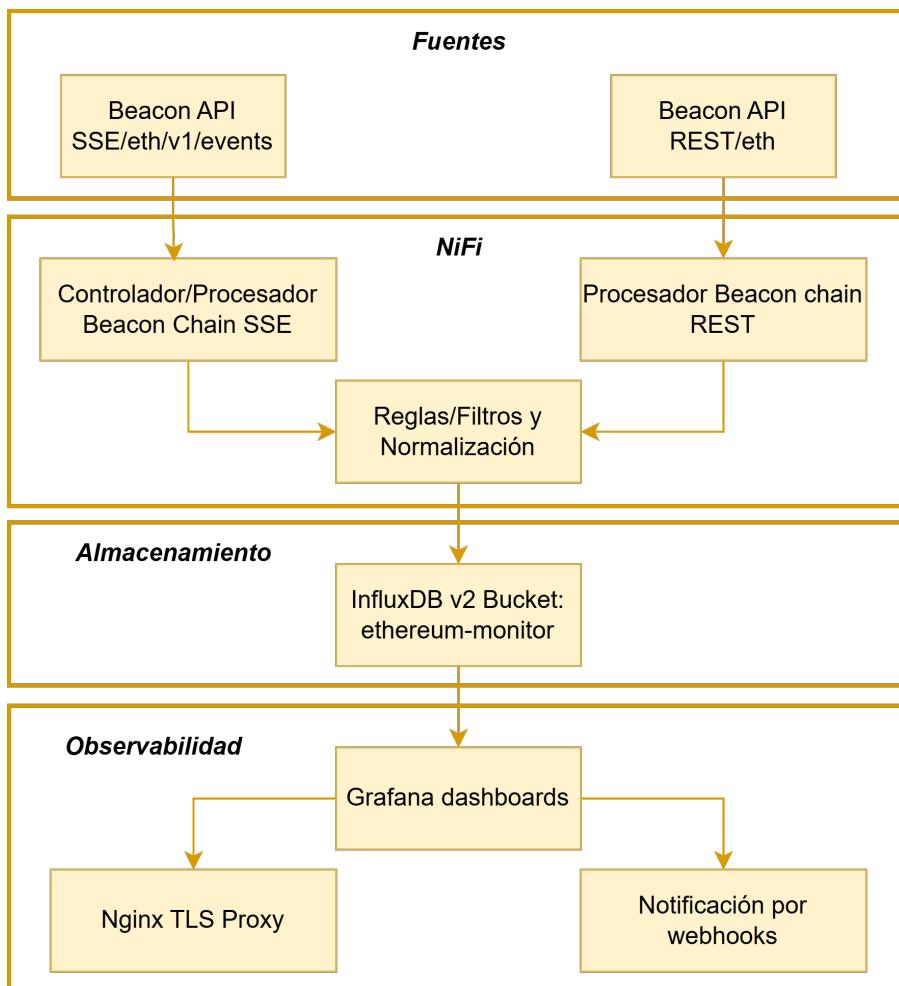


Figura 3.1: Arquitectura general del sistema de monitoreo que muestra el flujo de datos desde la Beacon API hacia NiFi, su almacenamiento en InfluxDB y la visualización y notificaciones a través de Grafana.

3.3. Visión en detalle del proyecto

3.3.1. Fuente de Datos: Nodo Ethereum y Beacon Chain API

Ethereum Beacon Chain API

La Beacon Chain API define el estándar para acceder a los datos de la capa de consenso de Ethereum. Su especificación, mantenida por la Ethereum Foundation [7], describe un conjunto de endpoints REST y canales SSE (Server Sent Events) que permiten obtener información sobre bloques, estados, validadores y eventos del consenso.

A diferencia de las interfaces RPC tradicionales, esta API utiliza exclusivamente HTTP-/HTTPS y JSON, lo que facilita su integración con sistemas modernos de observabilidad. Su diseño modular permite:

- Consultar datos históricos y de estado actual mediante REST (pull model).
- Recibir notificaciones en tiempo real mediante SSE (push model).
- Acceder a métricas críticas del consenso sin necesidad de ejecutar un nodo local.

Categorías de endpoints

La Beacon API se organiza en cuatro familias de información principales. Cada grupo presenta una frecuencia y una utilidad diferente en el contexto de monitoreo. Para mas detalle ir a la referencia:

Datos de Bloques: estructura, propósito y utilidad en el monitoreo El análisis de los bloques de la Beacon Chain constituye el núcleo del monitoreo sobre la red Ethereum, ya que cada bloque agrupa tanto la información estructural del consenso como los datos económicos de la capa de ejecución.

A partir de la combinación de múltiples endpoints de la Beacon Node API, se recopilan métricas que describen la actividad, estabilidad y rendimiento del sistema en tiempo real.

Objetivo de la extracción El objetivo general es obtener una visión temporal y estructural de la red, comprendiendo cómo se comporta el consenso, qué volumen de operaciones se procesa en cada bloque y cómo evolucionan los mecanismos de incentivos y escalabilidad.

Los datos se utilizan para:

- Medir la continuidad del consenso y la velocidad con que los bloques son propuestos y finalizados.
- Evaluar la actividad económica, observando depósitos, retiros y recompensas de los validadores.

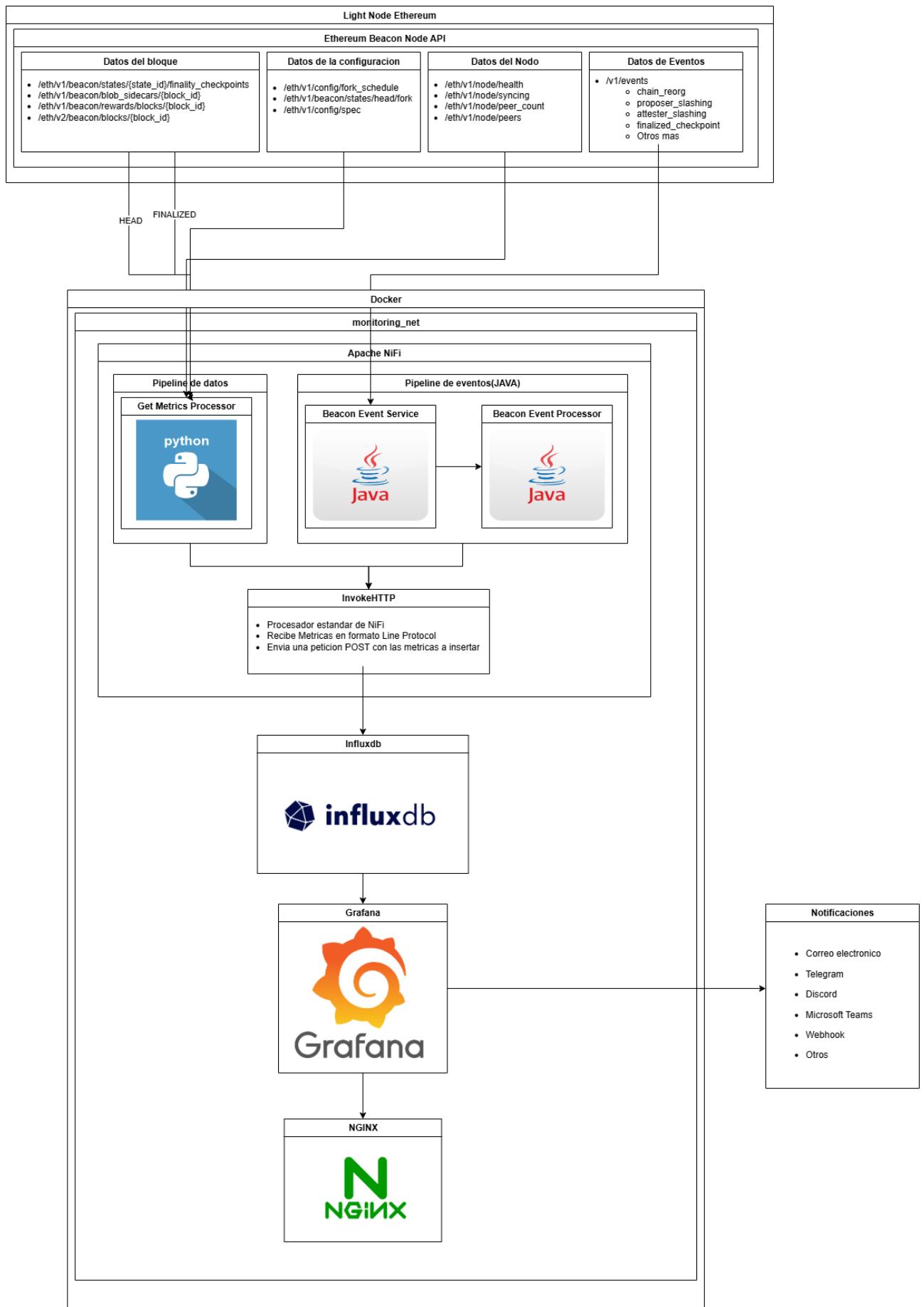


Figura 3.2: Arquitectura detallada del sistema de monitoreo propuesto para Ethereum

Ethereum Beacon Node API			
Datos del bloque	Datos de la configuración	Datos del Nodo	Datos de Eventos
<ul style="list-style-type: none"> • /eth/v1/beacon/states/{state_id}/finality_checkpoints • /eth/v1/beacon/blob_sidecars/{block_id} • /eth/v1/beacon/rewards/blocks/{block_id} • /eth/v2/beacon(blocks/{block_id}) 	<ul style="list-style-type: none"> • /eth/v1/config/fork_schedule • /eth/v1/beacon/states/head/fork • /eth/v1/config/spec 	<ul style="list-style-type: none"> • /eth/v1/node/health • /eth/v1/node/syncing • /eth/v1/node/peer_count • /eth/v1/node/peers 	<ul style="list-style-type: none"> • /v1/events <ul style="list-style-type: none"> ◦ chain_reorg ◦ proposer_slashing ◦ attester_slashing ◦ finalized_checkpoint ◦ Otros mas

Figura 3.3: Vista general de los tipos de datos expuestos por un light node de Ethereum mediante la Beacon Node API, incluyendo bloques, configuración, estado del nodo y eventos.

- Analizar la eficiencia de la capa de ejecución, mediante métricas de gas y volumen transaccional.
- Monitorear innovaciones de escalabilidad, como los blobs introducidos en la EIP-4844.

Estos datos son procesados en los flujos de extracción, aplanados y luego almacenados en InfluxDB, generando series temporales que se visualizan en Grafana a través de paneles comparativos entre los estados *head* y *finalized*.

Ejemplo 1 – Endpoint /eth/v2/beacon/blocks/{block_id} Permite acceder a la información completa de un bloque, integrando:

- Datos de consenso (slot, proposer_index, raíces de estado y bloque padre).
- Datos de ejecución (gas, transacciones, retiros y base_fee).
- Listas de operaciones como attestations, deposits, withdrawals y slashings.

A partir de esta información se generan métricas derivadas —como el conteo de operaciones o la suma total de depósitos— que sirven para evaluar la actividad y complejidad de cada bloque.

El seguimiento periódico de los bloques head y finalized permite detectar retrasos o desincronización entre nodos, reflejando el estado operativo del consenso y el rendimiento de la red en tiempo real.

Ejemplo 2 – Endpoint /eth/v1/beacon/blob_sidecars/{block_id} Este endpoint incorpora la dimensión de escalabilidad y datos extendidos, introducida con la EIP-4844 (Proto-Danksharding).

Permite obtener los blob sidecars asociados a cada bloque, que contienen los datos binarios utilizados por soluciones de rollups para reducir costos de almacenamiento.

El monitoreo de la cantidad de blobs por bloque ofrece información sobre:

- La adopción del modelo de datos extendidos y el nivel de carga de la red.

- La consistencia entre bloques head y finalized, detectando pérdidas o retrasos en la propagación de blobs.
- La eficiencia del consenso durante la incorporación de nuevas versiones del protocolo.

Datos de la configuración de la red Los endpoints de configuración de la Beacon Node API proporcionan información estática y de contexto sobre el entorno operativo del nodo. A diferencia de los endpoints dinámicos de bloques o eventos, estos exponen los parámetros fundamentales del protocolo de consenso, las versiones activas de red y la relación entre la capa de consenso (Beacon Chain) y la capa de ejecución (Ethereum).

Su propósito en el sistema de monitoreo es garantizar que el nodo consultado esté correctamente alineado con la configuración de la red, especialmente durante actualizaciones de protocolo o bifurcaciones (forks).

La información recolectada se utiliza para:

- Verificar coherencia entre nodos, asegurando que todos operen bajo la misma versión del consenso.
- Contextualizar métricas de rendimiento o consenso, asociándolas a la versión activa del protocolo (por ejemplo, Capella, Deneb o Electra).
- Detectar errores de configuración o versiones obsoletas, que podrían afectar la compatibilidad del monitoreo o la validez de los datos.

Estas métricas se almacenan en InfluxDB bajo la medición beacon_config_info y son utilizadas en los paneles de Grafana para mostrar el estado de configuración del nodo, la versión de consenso activa y la información del contrato de depósitos asociado a la red principal.

Ejemplo 1 – Endpoint /eth/v1/config/fork_schedule Este endpoint devuelve la programación de bifurcaciones (forks) conocidas por el nodo Beacon.

Cada fork representa una transición de versión del protocolo de consenso, introducida mediante una actualización de red (por ejemplo, Altair, Bellatrix, Capella o Deneb).

La respuesta contiene una lista de objetos con:

- previous_version: versión anterior al fork.
- current_version: versión actual o entrante.
- epoch: número del epoch en el que se activa la nueva versión.

El monitoreo de esta información permite:

- Confirmar la alineación del nodo con la versión actual de la red.

- Detectar forks inminentes, lo cual resulta clave para la planificación de actualizaciones de clientes o análisis de cambios en las métricas de rendimiento.
- Relacionar eventos históricos con cambios de versión, contextualizando comportamientos de red antes y después de un fork.

En Grafana, estos datos se representan para indicar la versión de consenso activa (por ejemplo, Deneb) y el número de epoch donde se produjo la transición.

Ejemplo 2 – Endpoint /eth/v1/config/deposit_contract Este endpoint devuelve la dirección del contrato de depósitos de la capa de ejecución y el identificador de cadena (chain_id) correspondiente.

Dicha información es fundamental para establecer la conexión entre la Beacon Chain (Eth2) y la capa de ejecución (Eth1), dado que los depósitos realizados en este contrato activan nuevos validadores en la red.

Los campos principales son:

- chain_id: identificador de red (por ejemplo, 1 para Mainnet).
- address: dirección del contrato de depósitos activo.

El monitoreo de este endpoint permite:

- Verificar que el nodo observa la dirección de contrato correcta, evitando errores de configuración en redes de prueba o entornos locales.
- Asegurar la coherencia entre capa de ejecución y consenso, garantizando que las métricas extraídas correspondan a la misma red.
- Correlacionar la versión del consenso (Eth-Consensus-Version) con el contrato asociado, validando compatibilidad entre la configuración del nodo y la red que sigue.

Datos del nodo Los endpoints del grupo `/eth/v1/node/*` ofrecen una visión directa del estado operativo, la conectividad y la sincronización del cliente Beacon dentro de la red Ethereum.

Mientras que los endpoints de bloques y consenso permiten observar el comportamiento de la cadena, estos se centran en la salud del nodo: su integración con otros pares (peers), su grado de sincronización con la red y su disponibilidad general.

Su propósito en el sistema de monitoreo es evaluar la estabilidad, propagación y disponibilidad del nodo, detectando posibles anomalías que afecten la precisión o integridad de los datos recolectados.

Los valores obtenidos permiten determinar si el nodo:

- Está correctamente sincronizado con la red.

- Mantiene una cantidad suficiente de conexiones P2P activas.
- Puede responder de forma estable a solicitudes externas o balanceadores.

En el sistema implementado, estas métricas se registran en InfluxDB bajo mediciones como beacon_node_status o beacon_peer_info, y son visualizadas en Grafana para proporcionar indicadores de disponibilidad, número de peers y estado de sincronización en tiempo real.

Ejemplo 1 – Endpoint /eth/v1/node/peers Este endpoint devuelve la lista de peers conectados al nodo Beacon y su estado de conexión actual.

Permite conocer tanto las conexiones activas como las que están en proceso de conexión, desconexión o pendientes.

Cada peer incluye información como:

- peer_id: identificador único del nodo remoto.
- last_seen_p2p_address: dirección IP y puerto de contacto.
- state: estado de conexión (connected, disconnected, etc.).
- direction: dirección de conexión (inbound o outbound).

Estos datos son esenciales para evaluar la salud de la red P2P.

Un número bajo de conexiones o una alta tasa de desconexiones puede indicar:

- Problemas de configuración o aislamiento de red.
- Limitaciones en las conexiones entrantes o salientes.
- Desbalances en la propagación de bloques y atestaciones.

El conteo total de peers (campo meta.count) se utiliza para generar métricas de disponibilidad y topología, que permiten medir el grado de interacción del nodo con el resto de la red Ethereum.

Ejemplo 2 – Endpoint /eth/v1/node syncing Este endpoint indica el estado de sincronización del nodo, es decir, si se encuentra actualizado con la cadena principal o aún está procesando bloques anteriores.

Proporciona campos como:

- head_slot: slot más reciente conocido por el nodo.
- sync_distance: cantidad de slots restantes para alcanzar el último bloque.
- is_syncing: indica si el nodo aún está sincronizando.

- `is_optimistic`: señala si la capa de ejecución aún no ha sido completamente verificada.
- `el_offline`: determina si la capa de ejecución está desconectada.

Estos datos son fundamentales para verificar la operatividad del nodo:

- Si `is_syncing` = true o `sync_distance` es elevado, puede haber retrasos significativos en la propagación de bloques.
- Si `el_offline` = true, el nodo puede no estar recibiendo información de la capa de ejecución, afectando la integridad de los datos.

En entornos de monitoreo, esta información se transforma en métricas de sincronización y disponibilidad, generando alertas cuando el nodo pierde conexión con la red o se queda rezagado respecto al head global.

Datos de eventos El endpoint `/eth/v1/events` de la Beacon Node REST API permite establecer una conexión en tiempo real con el flujo continuo de eventos generados por el nodo, utilizando el protocolo *Server-Sent Events* (SSE).

A diferencia de los endpoints tradicionales que devuelven datos bajo demanda, este canal transmite de manera persistente y automática cada cambio relevante que ocurre dentro de la *Beacon Chain*.

Su función principal es proveer visibilidad instantánea sobre las transiciones del consenso, como la aparición de nuevos bloques, cambios de cabeza de cadena (*head*), actualizaciones de finalización, penalizaciones de validadores (*slashing*) o reorganizaciones (*reorgs*).

Esta naturaleza reactiva convierte al endpoint en una pieza esencial para sistemas de monitoreo de alta disponibilidad, indexadores y plataformas de observabilidad que requieren alertas inmediatas ante anomalías en el consenso.

Estructura y parámetros El endpoint acepta el parámetro obligatorio `topics`, que define los tipos de eventos a suscribirse, tales como:

- `head` y `block`: notificaciones de nuevos bloques y actualizaciones de cabeza.
- `finalized_checkpoint`: eventos de finalización del consenso.
- `chain_reorg`: reorganizaciones de la cadena.
- `proposer_slashing` y `attester_slashing`: penalizaciones aplicadas a validadores.
- `blob_sidecar` y `payload_attributes`: eventos relacionados con la capa de ejecución y blobs (EIP-4844).

Cada mensaje SSE se envía con el formato estándar:

```
event: <nombre_del_evento> data: <payload_JSON>
```

y permanece activo mientras el canal siga abierto, transmitiendo los eventos en el momento en que ocurren.

Importancia en el monitoreo El endpoint `/eth/v1/events` constituye la columna vertebral de la observabilidad en tiempo real dentro de la red Ethereum.

Su capacidad de transmitir de manera inmediata las actualizaciones del consenso permite:

- Detectar en el instante la aparición de reorganizaciones de cadena (reorgs), que pueden indicar comportamientos anómalos de la red o discrepancias entre nodos.
- Identificar penalizaciones (slashing), tanto de proposers como de attesters, que revelan intentos de doble voto, propuestas conflictivas o incumplimientos del protocolo.
- Observar la frecuencia de bloques y transiciones de epoch, lo que refleja el ritmo de producción y la estabilidad del consenso.
- Correlacionar eventos en tiempo real con métricas de ejecución, facilitando el análisis conjunto de consenso y rendimiento.

Aplicación práctica en el sistema de monitoreo En el flujo de monitoreo implementado, el endpoint `/eth/v1/events` es consumido por un proceso Python persistente que escucha los eventos seleccionados —principalmente `head`, `chain_reorg`, `proposer_slashing` y `attester_slashing`.

Cada evento recibido se procesa inmediatamente y se transforma en una línea temporal con etiquetas de:

- Tipo de evento.
- Identificador del bloque o slot.
- Epoch.
- Versión de consenso activa (Eth-Consensus-Version).

Los datos se insertan en la base InfluxDB bajo la medición `beacon_realtime_events`, desde donde se visualizan en paneles de Grafana que muestran:

- La frecuencia de aparición de reorgs.
- Los eventos de slashing y sus validadores afectados.
- La correlación entre actividad de bloques y estabilidad de la red.

Relevancia para el diagnóstico y la confiabilidad El monitoreo de eventos en tiempo real cumple un papel crítico para la integridad del sistema de observabilidad, ya que:

- Un aumento de eventos `chain_reorg` puede indicar inestabilidad de consenso, fallos de sincronización o divergencia entre clientes.

- La presencia de proposer_slashing y attester_slashing permite identificar comportamientos deshonestos o errores de validadores, anticipando problemas que podrían comprometer la seguridad del consenso.
- La detección inmediata de finalized_checkpoint confirma el avance estable de la red y la ausencia de bifurcaciones prolongadas.

3.3.2. Arquitectura de despliegue con Docker Compose

La infraestructura del sistema se encuentra contenedorizada y orquestada mediante Docker Compose, lo que permite administrar múltiples servicios interconectados dentro de un entorno aislado, reproducible y fácilmente desplegable.

El archivo docker-compose.yml define todos los componentes necesarios para el funcionamiento del sistema de monitoreo, sus dependencias, redes internas, volúmenes persistentes y configuraciones de entorno.

El uso de Docker Compose aporta portabilidad, estandarización y simplicidad operativa, ya que cada servicio mantiene su propio contenedor, dependencias y configuración. Esto evita conflictos entre versiones, facilita la automatización del despliegue y reduce la complejidad de instalación manual.

Estructura del archivo docker-compose.yml El archivo docker-compose.yml se estructura en tres secciones principales:

1. services: define los contenedores que forman parte del sistema.
2. volumes: especifica los volúmenes persistentes que aseguran la conservación de datos y configuraciones.
3. networks: establece la red interna utilizada para la comunicación entre servicios.

Esta estructura garantiza un despliegue modular donde cada componente puede iniciarse, detenerse o actualizarse de manera independiente, sin afectar al resto.

Configuración de red Todos los servicios definidos en el archivo comparten una red interna común llamada monitoring_net, configurada con el driver bridge.

Esta red proporciona aislamiento respecto a otros contenedores externos al sistema y permite que los servicios se comuniquen utilizando nombres lógicos (por ejemplo, influxdb, nifi, grafana) en lugar de direcciones IP.

El uso de una red interna dedicada facilita la comunicación directa entre contenedores, evita la exposición innecesaria de puertos al host y permite un control más granular sobre las dependencias y políticas de conexión.

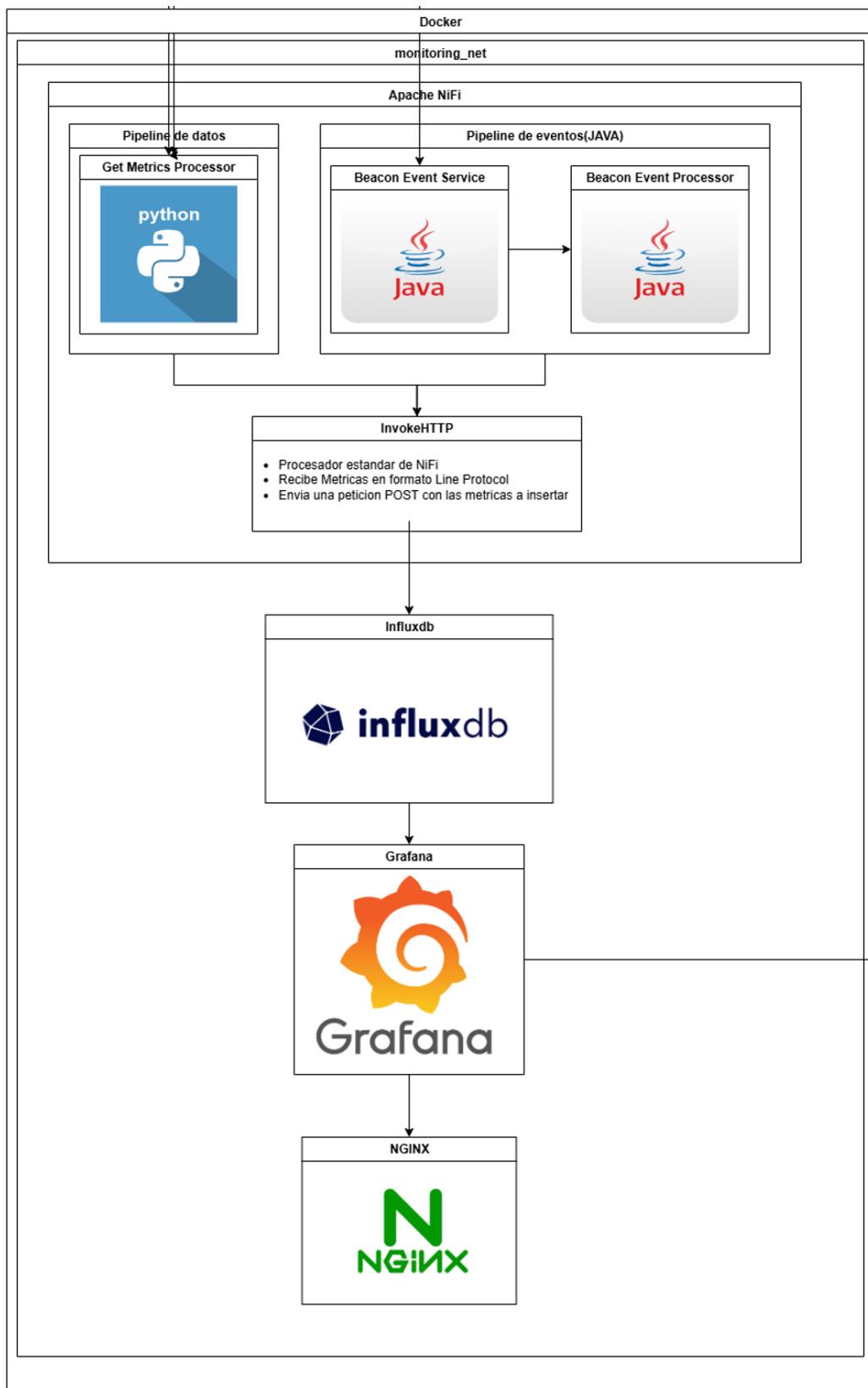


Figura 3.4: Flujo interno del procesamiento en Apache NiFi que envía métricas hacia InfluxDB para su posterior visualización en Grafana y publicación mediante NGINX.

Persistencia de datos El archivo define un volumen persistente llamado influxdb_data, que almacena los datos generados por InfluxDB en la ruta interna /var/lib/influxdb2.

Esto garantiza que los datos históricos, configuraciones y tokens de autenticación permanezcan intactos incluso si los contenedores se eliminan o reinician.

El uso de volúmenes también facilita la migración o respaldo del sistema, ya que la información crítica se mantiene fuera del ciclo de vida de los contenedores.

Variables de entorno y parametrización Cada servicio definido en el Compose utiliza variables de entorno \${VARIABLE} cargadas desde un archivo externo .env.

Este enfoque separa la configuración sensible del código, permitiendo modificar credenciales, tokens, puertos y zonas horarias sin alterar el archivo principal.

Ejemplos comunes de variables:

- Credenciales de usuario y contraseñas (NIFI_USER, INFLUXDB_USER, etc.).
- Tokens de autenticación (INFLUXDB_ADMIN_TOKEN).
- Identificadores de organización y buckets (INFLUXDB_ORG, INFLUXDB_BUCKET).
- Zona horaria global (TZ).

Esta práctica mejora la seguridad, la reutilización del archivo Compose y la facilidad para replicar entornos en distintos sistemas.

Control de dependencias y reinicio El uso de la instrucción depends_on garantiza que los servicios se inicien en el orden correcto.

Por ejemplo, NiFi y Grafana dependen de que InfluxDB esté disponible antes de iniciar, evitando errores de conexión durante la fase de arranque.

Adicionalmente, el parámetro restart: unless-stopped asegura que los contenedores se reinicien automáticamente en caso de fallo, manteniendo la alta disponibilidad del sistema.

Estas configuraciones hacen que el despliegue sea autosuficiente y resiliente, capaz de recuperarse sin intervención manual tras reinicios del host o cortes de energía.

1. InfluxDB → 2. NiFi → 3. Grafana → 4. NGINX

Beneficios del enfoque contenedorizado El uso de Docker Compose proporciona ventajas clave para este proyecto:

- Reproducibilidad: todo el entorno puede levantarse con un solo comando (docker compose up -d).
- Escalabilidad: se pueden añadir nuevos servicios o réplicas sin modificar el entorno base.
- Aislamiento: cada servicio ejecuta sus dependencias en un contenedor propio.

- Portabilidad: el mismo stack puede ejecutarse en entornos locales, servidores o nubes sin cambios.
- Mantenibilidad: facilita actualizaciones y limpieza de servicios mediante comandos simples (up, down, restart).

3.3.3. InfluxDB – Almacenamiento de series temporales

Rol dentro del sistema InfluxDB actúa como el núcleo de persistencia temporal del stack, almacenando todas las métricas obtenidas desde los endpoints de la Beacon Chain API.

Su elección se debe a su eficiencia para manejar grandes volúmenes de datos cronológicos y su compatibilidad directa con herramientas de observabilidad como Grafana.

Cada registro proveniente de Apache NiFi se inserta en InfluxDB mediante Line Protocol, un formato simple de escritura que define de forma explícita el measurement, las tags (metadatos indexables) y los fields (valores medidos), junto con su timestamp.

Este formato permite mantener un almacenamiento optimizado y consultas rápidas basadas en tiempo.

Estructura de almacenamiento Los datos se organizan dentro del bucket:

INFLUXDB_ORG=fpuna

INFLUXDB_BUCKET=ethereum-monitor

El bucket agrupa las mediciones provenientes de distintos endpoints, que se clasifican como:

- beacon_node_info → información del estado operativo del nodo .
- beacon_block_info → métricas de bloques.
- beacon_event → eventos transmitidos por el stream SSE.
- beacon_config_info → parámetros de configuración y fork schedule.

Cada medición incorpora etiquetas clave como:

- network → red a la que pertenece el nodo (ej. mainnet).
- source → origen o URL del nodo monitorizado (ej. <https://www.lightclientdata.org>).
- endpoint → API específica que originó el dato (por ejemplo /eth/v1/node/health).

Estas etiquetas permiten consultas multidimensionales que combinan temporalidad, procedencia y tipo de métrica.

Line Protocol y flujo de inserción NiFi se encarga de aplanar los objetos JSON obtenidos de los endpoints y convertirlos en registros individuales compatibles con Influx.

Cada registro resultante adopta el siguiente esquema general:

```
<measurement>,network=<network>,source=<url>,endpoint=<endpoint>
<field>=<value> <timestampl>
```

Ejemplo simplificado:

```
beacon_node_info, network=mainnet, source=data.org, endpoint=/eth/v1/node syncing
is_syncing=true, el_offline=false 1731952275000000000
```

Esta representación favorece la eficiencia en escritura y una estructura flexible, donde cada endpoint puede extenderse con nuevos campos sin alterar el esquema global.

Consultas y procesamiento en Flux El lenguaje Flux se utiliza para consultar los datos en Grafana.

Su sintaxis funcional facilita operaciones complejas sobre las series temporales, como agragaciones, mapeos y filtrados.

Ejemplos comunes dentro de los dashboards son:

Filtrado y selección de los valores de los últimos 5 minutos:

```
from(bucket: "ethereum-monitor")
|> range(start: -5m)
|> filter(fn: (r) => r["_measurement"] == "beacon_node_info")
|> filter(fn: (r) => r["_field"] == "is_syncing")
|> last()
```

Estas funciones permiten que Grafana pueda mostrar tanto indicadores instantáneos (como status codes o flags de sincronización) como tendencias históricas (por ejemplo, evolución de slots o cantidad de peers).

Ventajas en el contexto del proyecto

- Bajo consumo de recursos y alta tasa de escritura.
- Modelo de datos flexible, sin necesidad de esquemas fijos ni migraciones.
- Integración nativa con Grafana, facilitando el desarrollo de dashboards interactivos.
- Soporte de múltiples retenciones y buckets, lo que permite gestionar diferentes horizontes de tiempo si se requiere escalar la solución.

3.3.4. Grafana – Visualización y análisis de métricas

Rol dentro del sistema Grafana representa la capa de visualización del stack, proporcionando una interfaz gráfica dinámica para explorar y analizar los datos almacenados en InfluxDB. Su función principal es transformar las métricas y eventos recolectados por NiFi en paneles interactivos que permiten al usuario interpretar el comportamiento de la red Ethereum y el estado del nodo en tiempo real.

Gracias a su arquitectura extensible, Grafana se integra directamente con InfluxDB mediante su plugin nativo, permitiendo ejecutar consultas en Flux y representar los resultados en múltiples formatos: gráficas temporales, tablas, indicadores y paneles estadísticos.

Esta capacidad lo convierte en una herramienta ideal para construir un entorno de observabilidad modular y autogestionado.

Conexión con InfluxDB Grafana utiliza un datasource basado en el bucket: ethereum-monitor, perteneciente a la organización: fpuna, donde se almacenan los datos enviados desde NiFi.

La conexión se configura a través del panel de provisioning, con las credenciales y token definidos en las variables de entorno:

```
INFLUXDB_ORG=${INFLUXDB_ORG}  
INFLUXDB_BUCKET=${INFLUXDB_BUCKET}  
INFLUXDB_ADMIN_TOKEN=${INFLUXDB_ADMIN_TOKEN}
```

Una vez establecida la conexión, Grafana puede consultar directamente los measurements (por ejemplo, beacon_event, beacon_node_info, beacon_block_info, beacon_config_info) aplicando filtros por etiquetas (network, source, endpoint) y transformaciones en tiempo real.

Estructura de paneles Cada panel de Grafana se asocia a una consulta Flux específica que define:

- Measurement: la métrica o conjunto de métricas a analizar.
- Filtros: condiciones que limitan el origen de los datos, como red o endpoint.
- Ventana temporal (aggregateWindow): define la resolución temporal de los datos visualizados.
- Transformaciones (map, filter, yield): ajustan el formato o la escala de los valores antes de graficarlos.

Los paneles se agrupan en dashboards temáticos (por ejemplo, Beacon Config, Block Metrics, Eventos), permitiendo una visión segmentada del sistema.

Cada dashboard está configurado con variables globales (source, network, interval, block_id) que posibilitan el filtrado y actualización simultánea de todos los paneles.

Características visuales y operativas

- Actualización automática: la mayoría de los dashboards se refrescan cada 5–30 segundos, garantizando que los gráficos reflejen el estado actual del nodo.
- Visualizaciones múltiples: series temporales, tablas y paneles “stat” se combinan para ofrecer vistas numéricas, históricas y comparativas.
- Temas dinámicos y acceso anónimo: el entorno permite habilitar el modo viewer sin autenticación (GF_AUTH_ANONYMOUS_ENABLED=true), útil para mostrar métricas en contextos públicos o académicos.
- Healthcheck integrado: se define un endpoint de supervisión (/api/health) para monitorear la disponibilidad de Grafana, usado en los scripts de arranque del contenedor y en Telegraf.

Propósito dentro del proyecto Dentro del ecosistema de monitoreo de Ethereum, Grafana cumple la función de observatorio unificado, donde convergen:

- Los valores de configuración de la red (specs, forks).
- Las métricas operativas del nodo (sincronización, peers, estado).
- Los parámetros de bloque (gas, transacciones, recompensas, blobs).
- Los eventos de consenso recibidos en streaming SSE.

Este entorno proporciona una representación integral del flujo de datos desde la capa de consenso, permitiendo validar que el sistema cumple con los objetivos de observabilidad y detección de anomalías en tiempo real.

Dashboard: Beacon Node Status

Propósito El dashboard Beacon Node Status ofrece una visión general del estado operativo del nodo Ethereum Beacon, permitiendo evaluar su sincronización, conectividad y disponibilidad en tiempo real.

A través de este panel, es posible conocer si el nodo está correctamente alineado con la red, si mantiene conexión estable con la capa de ejecución, y si participa activamente dentro de la red P2P.

Su actualización frecuente proporciona una lectura continua de la salud y rendimiento del nodo, convirtiéndose en una herramienta esencial para garantizar la estabilidad del entorno de monitoreo.



Figura 3.5: Dashboard Beacon Node Status que muestra el estado operativo del nodo Ethereum Beacon.

Estado general del nodo El dashboard presenta indicadores visuales que reflejan si el nodo se encuentra en funcionamiento, si está sincronizando o si presenta fallas operativas. Esto permite detectar rápidamente problemas de disponibilidad, caídas del cliente o interrupciones en la conexión con la capa de ejecución.

Conexión con la capa de ejecución (Execution Layer) Muestra si el nodo mantiene una comunicación activa con su cliente de ejecución (por ejemplo, Geth o Nethermind). La pérdida de conexión con esta capa puede indicar un error crítico en la integración o un proceso detenido, por lo que este indicador resulta vital para validar el correcto acoplamiento tras The Merge.

Modo de sincronización y validación de datos Incluye métricas que muestran si el nodo está completamente sincronizado o si continúa descargando bloques. También se indica si se encuentra en modo optimista, es decir, si está proveyendo datos aún no completamente validados por la capa de ejecución. Estos indicadores son esenciales para confirmar que el nodo ofrece información confiable y finalizada.

Actividad de red (Peers y conectividad) El dashboard muestra la cantidad de pares conectados, conectándose o desconectándose, proporcionando una visión clara del estado de la red P2P del nodo. Esto permite identificar problemas de conectividad, aislamiento o inestabilidad en la comunicación con otros validadores.

Evolución temporal y progreso de sincronización Se representan series temporales que muestran el avance del número de slot y la distancia de sincronización con respecto a la cabeza de la cadena. Un progreso constante en estas métricas indica un nodo activo y estable;

en cambio, interrupciones o fluctuaciones anómalas pueden señalar cuellos de botella de red o errores en la base de datos local.

Valor en el monitoreo El dashboard Beacon Node Status cumple una función central dentro del sistema de observabilidad, al ofrecer una evaluación inmediata del estado del nodo y su desempeño dentro del consenso.

Su utilidad práctica incluye:

- Detectar fallos de sincronización o conexión antes de que afecten la recopilación de datos o el monitoreo de bloques.
- Evaluar la estabilidad P2P mediante el número y comportamiento de los peers.
- Verificar la comunicación entre capas de consenso y ejecución, asegurando la validez de los datos procesados.
- Identificar degradaciones de rendimiento o períodos de inactividad que podrían afectar la disponibilidad del nodo.

En conjunto, este panel proporciona una visión consolidada del funcionamiento interno del nodo, sirviendo como referencia principal para el diagnóstico y la supervisión continua de la infraestructura Ethereum en el entorno de monitoreo.

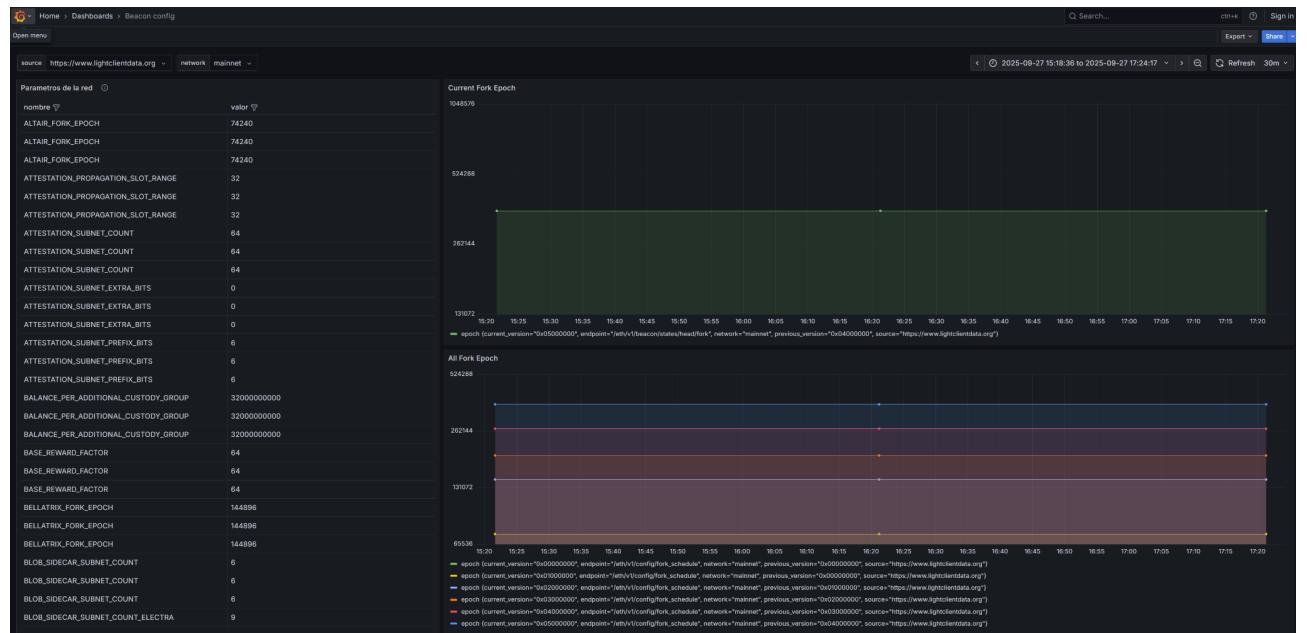


Figura 3.6: Dashboard Beacon Config que muestra la configuración del protocolo y el historial de forks de la red Ethereum.

Dashboard: Beacon Config

Propósito El dashboard Beacon Config ofrece una visión estructural de la configuración actual de la red Ethereum, permitiendo analizar los parámetros que definen el comportamiento del protocolo de consenso y la evolución histórica de sus bifurcaciones (forks).

A diferencia de los paneles centrados en métricas operativas o de rendimiento, este dashboard proporciona una perspectiva normativa y de contexto, ayudando a comprender en qué etapa del ciclo de actualizaciones se encuentra la red y bajo qué reglas opera el nodo monitoreado.

Información mostrada y utilidad

Parámetros del protocolo El dashboard muestra los valores que definen las reglas internas de la red Ethereum, como la duración de los slots, el número de slots por epoch, los límites de depósito y los coeficientes de penalización. Estos datos permiten verificar que el nodo utiliza la configuración correcta para la red a la que pertenece (por ejemplo, mainnet o holesky) y detectar posibles inconsistencias o desalineaciones de parámetros entre clientes o versiones. De esta forma, sirve como una herramienta de control para garantizar la coherencia del entorno de consenso.

Fork actual del protocolo El panel presenta el fork actualmente activo en el nodo, representando visualmente el momento de la red en relación con su versión del protocolo. Un valor estable indica que la red se mantiene en una misma versión, mientras que un cambio o salto visible en la serie temporal refleja la activación de una actualización del consenso. Esta información es clave para contextualizar eventos y métricas de otros dashboards, especialmente durante o después de actualizaciones importantes.

Historial y cronología de forks También se visualiza la secuencia completa de forks que la red ha atravesado, junto con los que están programados para activarse en el futuro. Esto permite entender la evolución del protocolo Ethereum a lo largo del tiempo y anticipar los cambios estructurales que puedan impactar en el comportamiento del nodo o del sistema de monitoreo. Cada punto del gráfico representa una transición en las reglas del consenso —por ejemplo, el paso entre Bellatrix, Capella o Deneb—, ofreciendo un marco histórico que puede correlacionarse con variaciones observadas en otras métricas.

Diseño y visualización El diseño del dashboard combina una tabla de configuración estática con gráficos de evolución temporal, organizados de forma que permitan una lectura rápida y contextual:

- La tabla resume los parámetros técnicos vigentes del protocolo.
- Los gráficos muestran tanto el fork actual como la secuencia de bifurcaciones históricas y programadas.

- Las variables dinámicas de red, fuente de datos e intervalo temporal permiten reutilizar el dashboard para diferentes entornos o redes de prueba.

En conjunto, la visualización ofrece una fotografía completa de cómo está configurada la red y cómo ha evolucionado, aportando contexto valioso para interpretar métricas y eventos observados en otros paneles.

Valor en el monitoreo El dashboard Beacon Config complementa los paneles de estado y eventos al proporcionar el marco de referencia técnico del consenso.

Su utilidad principal en el monitoreo radica en que permite:

- Verificar la coherencia de configuración entre nodos o entornos de red.
- Auditar las versiones del protocolo y confirmar que el cliente del nodo se encuentra actualizado.
- Correlacionar cambios estructurales del protocolo (forks) con variaciones operativas detectadas en otros dashboards.
- Anticipar actualizaciones programadas, facilitando la preparación del entorno ante cambios de red.

En suma, este dashboard actúa como un panel de control del consenso, aportando una base contextual que permite interpretar con precisión los datos operativos y validar que el nodo Ethereum monitorizado se mantiene correctamente alineado con la evolución del protocolo.



Figura 3.7: Dashboard Block Metrics que muestra métricas detalladas sobre los bloques producidos en la Beacon Chain de Ethereum.

Dashboard: Block Metrics

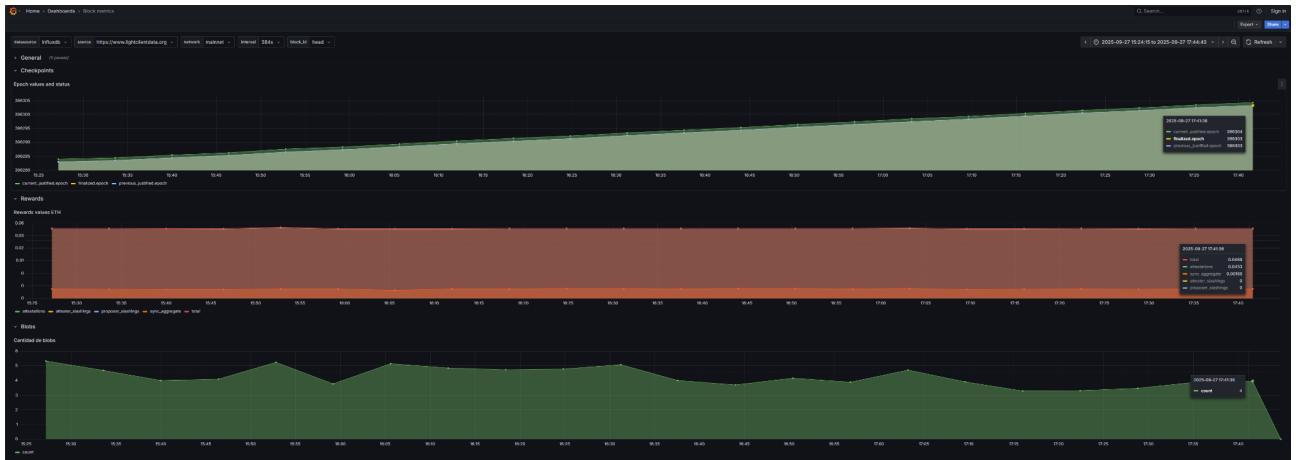


Figura 3.8: Dashboard Block Metrics (continuación) que muestra métricas detalladas sobre los bloques producidos en la Beacon Chain de Ethereum.

Propósito El dashboard Block Metrics tiene como finalidad monitorear de forma detallada el comportamiento y las características de los bloques producidos en la Beacon Chain de Ethereum.

A través de este panel, es posible analizar en tiempo real la evolución de la producción de bloques, su contenido, la eficiencia en el uso del gas y la actividad económica reflejada en depósitos, retiros, recompensas y eventos del consenso.

Este dashboard actúa como el núcleo analítico del sistema de observabilidad, ofreciendo una visión integral sobre el rendimiento del protocolo y la salud del proceso de validación.

Información mostrada y utilidad

Producción de bloques y slots El dashboard permite visualizar la secuencia de slots y números de bloque producidos por la red, representando el ritmo de generación de nuevos bloques en la cadena.

Un incremento constante en los valores indica un nodo sincronizado y en funcionamiento normal, mientras que interrupciones o caídas reflejan posibles fallas en la producción o propagación de bloques.

Esta sección constituye un indicador directo de la continuidad y estabilidad del consenso.

Eficiencia del gas y rendimiento de ejecución Se incluyen métricas relacionadas con el consumo de gas y la capacidad de procesamiento de la red, como base_fee, gas_used, gas_limit y blob_gas.

Estas métricas permiten evaluar la carga y congestión de la capa de ejecución, identificando picos de actividad o períodos de baja utilización.

Asimismo, su comparación temporal facilita detectar cambios en el comportamiento económico del bloque, especialmente tras actualizaciones o variaciones en la demanda de transacciones.

Actividad económica y volumen de operaciones El panel muestra sumas de valores asociados a los montos de depósitos, retiros y transferencias en cada bloque, expresados en ETH.

También presenta contadores de eventos como attestations, transactions, withdrawals, deposits o slashings, que reflejan el nivel de participación y complejidad de los bloques.

Estas métricas permiten identificar picos de actividad económica, validadores activos y posibles anomalías en la frecuencia de operaciones, sirviendo como referencia del movimiento interno de la red.

Progreso del consenso y checkpoints El dashboard incluye indicadores que muestran la evolución de los checkpoints justificados y finalizados, junto con sus respectivos epochs.

Estos valores son fundamentales para confirmar que la red está alcanzando finalización de forma constante y dentro de los intervalos esperados.

Un estancamiento o variación irregular puede señalar problemas de consenso o desincronización entre nodos validadores.

Recompensas del validador Las recompensas obtenidas por los validadores al proponer o participar en bloques se visualizan normalizadas a ETH.

Esto permite analizar el rendimiento económico del proceso de validación y detectar correlaciones entre la estabilidad del nodo, la participación y las recompensas recibidas.

Es un indicador directo del incentivo y desempeño de los validadores dentro del consenso.

Blobs y actividad de datos extendidos El dashboard incorpora métricas sobre la cantidad de blobs asociados a cada bloque, una funcionalidad introducida con la actualización EIP-4844 (Proto-Danksharding).

El seguimiento de estos valores permite evaluar el uso de espacio de datos auxiliar, indicador clave de la adopción de rollups y soluciones de escalado de capa 2.

Una mayor cantidad de blobs refleja un entorno más activo y con mayor demanda de procesamiento de datos.

Diseño y visualización El dashboard se organiza en secciones temáticas —General, Checkpoints, Rewards y Blobs— que agrupan las métricas por tipo de información.

Cada panel utiliza visualizaciones del tipo Time Series, lo que permite observar la evolución temporal de las métricas y comparar comportamientos entre distintos períodos.

La disposición del panel facilita un análisis progresivo, desde la producción de bloques y eficiencia del gas hasta las recompensas y el progreso del consenso.

Gracias a su diseño modular, puede adaptarse a diferentes redes o fuentes de datos manteniendo una estructura homogénea.

Valor en el monitoreo El Block Metrics es uno de los dashboards más importantes del sistema, ya que integra las dimensiones técnica, económica y de consenso del comportamiento de Ethereum.

Su valor en la observabilidad radica en que permite:

- Evaluar la estabilidad del proceso de validación y producción de bloques.
- Analizar la eficiencia y el consumo de recursos de la capa de ejecución.
- Correlacionar actividad económica y rendimiento del consenso, observando cómo varían las recompensas o los depósitos a lo largo del tiempo.
- Identificar anomalías operativas o eventos críticos, como irregularidades en gas, falta de finalización o picos de actividad inusual.

En conjunto, este dashboard proporciona una visión técnica y económica unificada del funcionamiento de los bloques en Ethereum, permitiendo detectar, contextualizar y explicar con precisión los comportamientos observados dentro del consenso.

Dashboard: Eventos

Propósito El dashboard Eventos permite monitorear en tiempo real la actividad de la red Ethereum desde la capa de consenso, mostrando la frecuencia y distribución de los principales eventos emitidos por la Beacon Chain.

A través de este panel, es posible visualizar cómo evoluciona el estado del nodo y de la red a medida que se producen nuevos bloques, validaciones o reorganizaciones.

Su objetivo principal es ofrecer una lectura continua del ritmo de actualización del consenso y del comportamiento colectivo de los validadores, aportando información crítica sobre la estabilidad y coherencia de la cadena.

Información mostrada y utilidad Eventos de validación (attestation)

Los eventos de tipo attestation representan los votos emitidos por los validadores para confirmar bloques dentro de un epoch.

El panel muestra su frecuencia como una serie temporal, donde una alta tasa de attestations indica una participación activa y saludable de los validadores en el consenso.

Variaciones anómalas o caídas abruptas en esta métrica pueden reflejar problemas de conectividad, latencia o falta de participación de validadores, afectando directamente la capacidad de finalización de la cadena.

En contextos de observabilidad, esta métrica permite:

- Verificar la regularidad de la participación de los validadores.
- Detectar retrasos o vacíos en la propagación de attestations.

- Correlacionar la actividad de validadores con el progreso de los checkpoints y la finalización de epochs.

Eventos de reorganización de cadena (reorg) Los eventos chain_reorg reflejan una situación donde el nodo abandona una rama anterior de la cadena para adoptar otra más reciente o con mayor justificación.

Aunque son eventos esperables en redes distribuidas, una frecuencia elevada o irregular puede ser un signo de inestabilidad o desacuerdo temporal entre nodos.

El panel permite visualizar cuándo ocurren estas reorganizaciones y con qué frecuencia, facilitando la detección de:

- Reorgs aislados, típicos de un comportamiento normal del consenso.
- Reorgs recurrentes o simultáneos, que pueden indicar fallas de sincronización, validadores desalineados o retrasos en la propagación de bloques.

Estos eventos son esenciales para evaluar la coherencia de la cadena en tiempo real, ya que cada reorganización implica una reescritura parcial del historial reciente, con potencial impacto en la validez de métricas dependientes del bloque head.

Otros tipos de eventos relevantes Además de los anteriores, el panel incluye otros tipos de actividad como:

- head: marca la llegada de un nuevo bloque cabeza.
- finalized_checkpoint: señala la finalización de un epoch.
- voluntary_exit o slashing: indican salidas o penalizaciones de validadores.

La representación simultánea de todos estos tipos de eventos permite obtener una visión integral del ciclo de consenso, desde la creación del bloque hasta su validación final.

Visualización El dashboard presenta un gráfico de series temporales donde cada tipo de evento se muestra con una línea distinta, lo que permite comparar la frecuencia relativa entre ellos.

La visualización está diseñada para operar en tiempo casi real, actualizándose cada pocos segundos, con una leyenda inferior que permite activar o desactivar categorías específicas.

Esta estructura facilita el análisis de patrones, como incrementos repentinos en attestations o la aparición de reorgs en momentos de alto tráfico o actualizaciones de red.

Valor en el monitoreo El dashboard Eventos cumple una función crítica dentro del sistema de observabilidad, ya que refleja el pulso del consenso en tiempo real.

Su valor reside en la capacidad de:

- Detectar reorgs inesperados que pueden comprometer la coherencia de la cadena.
- Monitorear la participación activa de los validadores mediante la observación de attestations.
- Identificar interrupciones o silencios de eventos, síntomas directos de desincronización o pérdida de conectividad.
- Correlacionar la actividad de la red con métricas de bloques, recompensas o finalización de epochs.

En conjunto, este dashboard actúa como un sensor de estabilidad del consenso, permitiendo anticipar comportamientos anómalos y validar que el nodo monitoreado mantiene una participación coherente y alineada con la red principal.

Nginx (Proxy y capa de exposición)

Propósito Nginx actúa como servidor web y proxy inverso, permitiendo publicar el stack de monitoreo de forma segura y controlada.

Su inclusión no es obligatoria para el funcionamiento interno del sistema, pero se vuelve esencial cuando se requiere acceso externo a los paneles de Grafana, endpoints de NiFi o APIs expuestas.

3.3.5. Apache NiFi (Ingesta y transformación de datos)

Propósito Apache NiFi es el componente encargado del flujo de datos (ETL) dentro del sistema de monitoreo.

Su función principal es extraer información desde los endpoints de la Beacon Chain API, transformarla en un formato estructurado y enviarla a InfluxDB para su almacenamiento temporal y análisis posterior.

Funciones principales

1. Ingesta de datos desde APIs

- Realiza consultas periódicas a los endpoints del nodo Beacon.
- Gestiona tanto respuestas estáticas como flujos continuos (Server-Sent Events desde /eth/v1/events).

2. Procesamiento y transformación

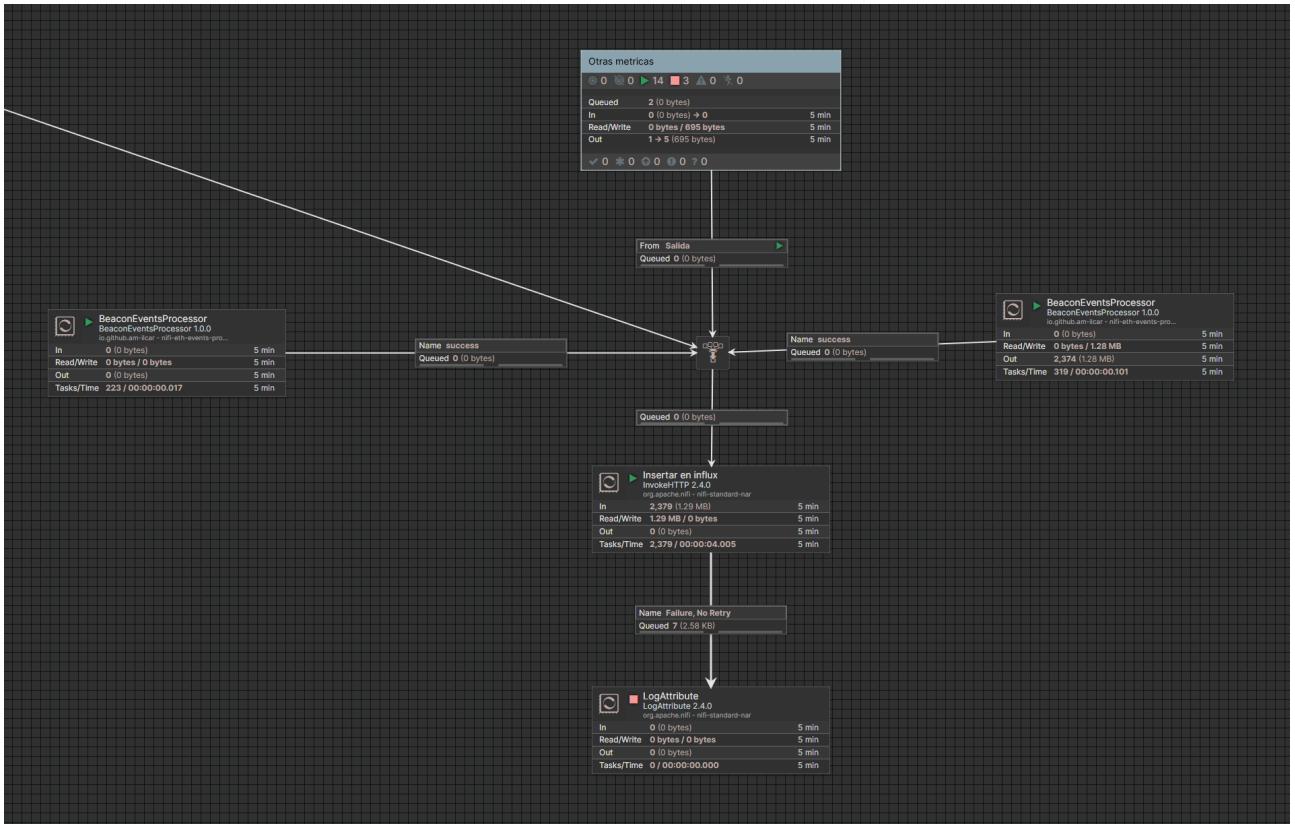


Figura 3.9: Canvas de Apache NiFi que muestra el flujo de ingestión y transformación de datos desde la Beacon Chain API hacia InfluxDB.

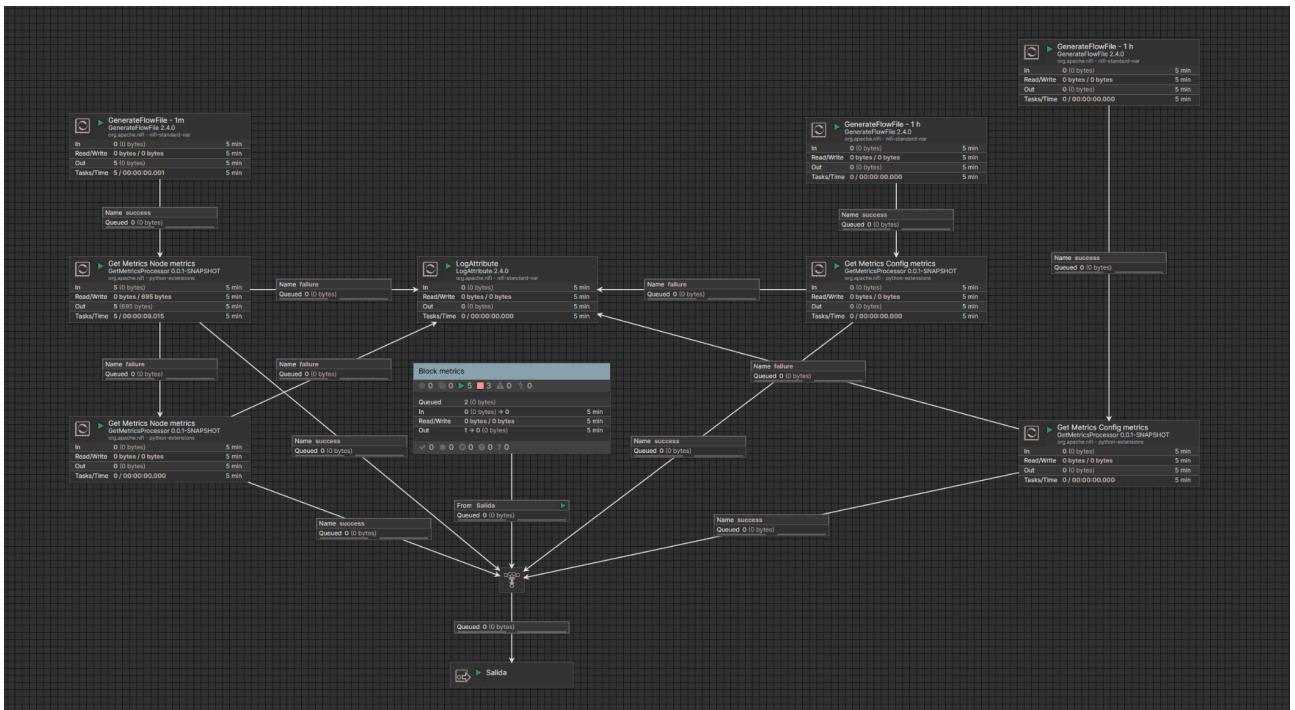


Figura 3.10: Canvas de Apache NiFi que muestra el flujo de ingestión y transformación de datos desde la Beacon Chain API hacia InfluxDB (continuación).

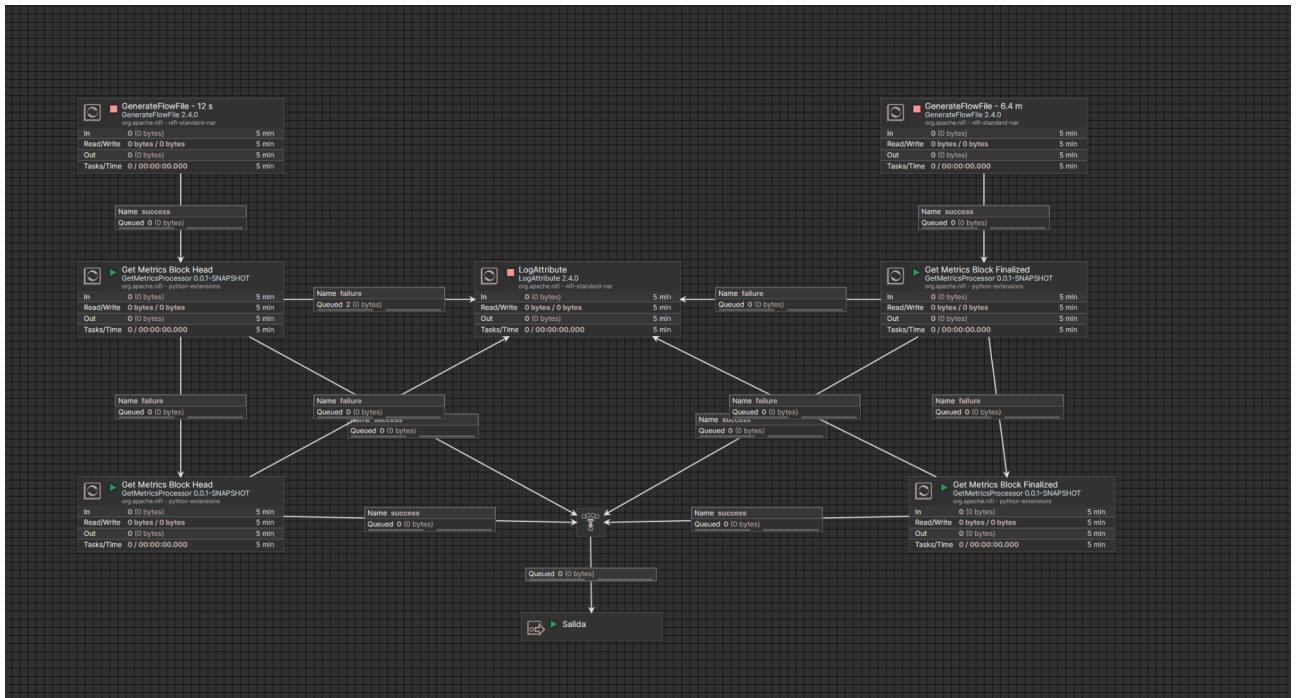


Figura 3.11: Canvas de Apache NiFi que muestra el flujo de ingesta y transformación de datos desde la Beacon Chain API hacia InfluxDB (continuación).

- Convierte las respuestas JSON en estructuras planas (flattened JSON).
- Aplica filtrados, asignación de campos y formatea la salida en Influx Line Protocol, compatible con la base de datos de series temporales.

3. Enrutamiento y control

- Utiliza procesadores de control de flujo (por ejemplo, RouteOnAttribute, MergeContent, PutInfluxDBRecord) para dirigir los datos según el tipo de endpoint o categoría (configuración, bloque, evento, etc.).
- Permite reintentos automáticos, colas internas y priorización de tareas.

4. Integración directa con InfluxDB

- Los datos ya transformados se envían directamente al bucket ethereum-monitor bajo diferentes measurements (por ejemplo, beacon_block_info, beacon_config_info).
- Esta conexión se realiza mediante un Parameter Context, que centraliza credenciales y configuración de destino.

3.3.6. Relación con los objetivos del trabajo

En sección 1.1.2 se formularon cinco objetivos específicos que orientan el presente trabajo. El contenido de este capítulo aporta evidencias concretas para cada uno de ellos, al pasar

de la descripción conceptual de Ethereum en Prueba de Participación a una arquitectura de monitoreo operativa y reproducible.

En relación con el primer objetivo, centrado en la elaboración de un marco conceptual actualizado, las secciones dedicadas a la Beacon Chain API, a los endpoints de bloques, configuración, estado de nodo y eventos muestran cómo los conceptos introducidos en el Capítulo 2 se traducen en métricas observables: qué se mide, con qué frecuencia y con qué finalidad dentro del contexto de la red post-*Merge*. El segundo objetivo, orientado al diseño de una arquitectura de referencia de ingesta, transformación, almacenamiento y visualización, se materializa aquí al detallar la cadena completa Beacon API → NiFi → InfluxDB → Grafana → NGINX, incluyendo tanto la vista lógica como la configuración de despliegue en contenedores.

Los objetivos tercero y cuarto, referidos al desarrollo de una librería *open source* para eventos SSE de la Beacon Chain y a la implementación de un *bundle* para NiFi orientado a eventos de consenso, se reflejan en la forma en que este capítulo presenta las fuentes de datos en tiempo real y el papel de NiFi como orquestador de la ingesta. Aquí se describe cómo estas herramientas se integran en el stack de monitoreo; los detalles de diseño, implementación y pruebas se abordan en profundidad en el Capítulo 4. Finalmente, el quinto objetivo, vinculado a la construcción de *dashboards* y reglas de alerta, encuentra apoyo en la descripción de los paneles Beacon Node Status, Beacon Config, Block Metrics y relacionados, que ilustran cómo la información recolectada se organiza para respaldar tanto la operación diaria como el análisis de incidencias.

En conjunto, esta sección actúa como puente entre los objetivos declarados y las herramientas concretas desarrolladas en los capítulos siguientes, mostrando de qué manera la arquitectura propuesta satisface los requisitos de observabilidad, reproducibilidad y apertura planteados para el sistema de monitoreo de Ethereum.

3.4. Características del entorno y configuración experimental

El stack propuesto se desplegó íntegramente con Docker Compose sobre un host de laboratorio dedicado a pruebas. La configuración se mantuvo deliberadamente simple para favorecer la reproducibilidad y dejar trazable cada dependencia incluida en el proyecto.

3.4.1. Plataforma de ejecución

La plataforma de ejecución correspondió a un host con 8 vCPU, 16 GB de RAM y disco SSD NVMe, ejecutando Linux (Ubuntu 22.04 LTS) con Docker Engine 25 y Docker Compose v2.24.0. La zona horaria del sistema se fijó en `TZ=America/Asuncion` para alinear las marcas de tiempo de InfluxDB con la recolección de datos.

Sobre este host se definió una red interna dedicada, `monitoring_net`, utilizando el driver

`bridge` y reservada exclusivamente para los contenedores del stack. Únicamente se expusieron hacia el host los puertos necesarios: 8086 para InfluxDB, 3000/443 para el acceso a Grafana a través de NGINX y, en el caso de NiFi, 8443 para la consola de administración y 8081 para la ingestión desde el cliente SSE.

En este entorno se desplegaron NiFi 2.4.0 (con los NAR personalizados), InfluxDB 2.7.11, Grafana 12.0.0, Telegraf 1.34.4 y NGINX (imagen *alpine*). La política de retención del bucket principal de métricas se configuró en cuatro semanas (`DOCKER_INFLUXDB_INIT_RETENTION=4w`), suficiente para el horizonte de análisis planteado en el trabajo.

3.4.2. Clientes de la Beacon API y fuentes de datos

La obtención de datos se realizó íntegramente sobre la red principal de Ethereum, consumiendo la Beacon API desde dos fuentes diferentes: <https://www.lightclientdata.org> y <https://eth2-beacon-mainnet.nodereal.io>. Estas URLs se declararon como endpoints de origen en la configuración del cliente SSE, que se suscribió simultáneamente a los tópicos `head`, `block`, `finalized_checkpoint`, `chain_reorg`, `proposer_slashing`, `attester_slashing`, `blob_sidecar` y a eventos de clientes ligeros.

La persistencia se llevó a cabo en InfluxDB, utilizando un bucket denominado `ethereum-monitor` dentro de la organización `fpuna` y empleando el formato Line Protocol con precisión en nanosegundos. Sobre estas series temporales se apoyan los dashboards de Grafana, aprovisionados mediante ficheros JSON de *provisioning* incluidos en el proyecto, lo que facilita la reconstrucción del entorno visual a partir de la misma configuración.

3.4.3. Ventana de pruebas instrumentada

La evaluación del sistema se realizó mediante un periodo continuo de 24 horas sobre red mainnet, equivalente a 7 200 slots y 225 epochs. Durante este periodo se verificó la completitud de los flujos de bloques `head` y `finalized`, así como la eventual aparición de eventos de reorganización de cadena o penalizaciones de validadores, con el fin de comprobar que el stack era capaz de capturar y registrar dichas situaciones.

Se recopilaron métricas de salud de InfluxDB, Grafana y NiFi en un bucket interno específico (`_monitoring`) con Telegraf. La ejecución del entorno se automatizó mediante ficheros de configuración de Docker Compose y variables de entorno, de forma que el experimento pueda reproducirse íntegramente con un único comando de despliegue sobre una plataforma compatible.

3.5. Resultados experimentales del sistema de monitoreo

El stack se ejecutó en el entorno detallado en sección 3.4.

3.5.1. Procedimiento experimental e instrumentación

Para estimar la latencia entre la generación de un evento y su visualización en los paneles, se comparó la marca de tiempo incluida en los mensajes SSE con el instante en que el mismo evento quedaba almacenado en InfluxDB y aparecía en Grafana. A lo largo del día se tomaron muestras periódicas de esa diferencia y se calcularon valores típicos (mediana) y máximos observados, que sirven de base para los rangos de latencia reportados más adelante.

La ausencia de pérdidas o duplicaciones en los tópicos más importantes (`head` y `finalized_checkpoint`) se verificó contrastando el número de *slots* y *epochs* esperados en 24 horas con los efectivamente registrados en las series temporales. En paralelo, se observaron las métricas básicas de CPU y memoria del host mediante el propio stack de monitoreo, y se realizaron reinicios controlados de servicios internos, así como cortes breves en una de las fuentes de la Beacon API, con el fin de comprobar que las colas de NiFi y las series almacenadas en InfluxDB se recuperaban sin huecos apreciables. Esta instrumentación ligera fue suficiente para respaldar cuantitativamente los resultados resumidos en la tabla 3.1, donde se resume los principales escenarios de prueba considerados, los objetivos perseguidos en cada caso, las métricas observadas y el resultado alcanzado.

En términos de volumen, la combinación de consultas periódicas a los endpoints REST y el consumo continuo del stream SSE generó del orden de decenas de miles de puntos de medición diarios. Las consultas de bloques *head* cada 12 segundos y de bloques *finalized* cada 384 segundos aportaron aproximadamente 30 000 líneas diarias, distribuidas entre métricas estructurales del bloque, checkpoints de finalización, blobs asociados y recompensas de validadores. A ello se suman las lecturas de salud de nodo y configuración de red, con una cadencia de entre un minuto y una hora según el endpoint, que completan el panorama operativo del cliente Beacon y de los parámetros de consenso vigentes.

El aporte del canal SSE fue más significativo en términos de eventos, pero no comprometió la estabilidad del sistema. La suscripción concurrente a tópicos como `head`, `block`, `finalized_checkpoint` y `blob_sidecar` produjo un flujo cercano a un evento por *slot* y por fuente, complementado por mensajes de mayor frecuencia vinculados a `attestation`, contribuciones y actualizaciones de clientes ligeros. Tras la aplanación de los objetos JSON y la supresión de campos binarios o listas de gran tamaño, el volumen final almacenado se mantuvo en el orden de 10^5 – 10^6 puntos diarios, sin saturar la capacidad de escritura de InfluxDB.

Desde el punto de vista operativo, las mediciones recolectadas por Telegraf muestran que el sistema se comportó de manera estable durante toda la campaña. Sobre un host de 8 vCPU y 16 GB de RAM, el conjunto de servicios (NiFi, InfluxDB, Grafana, NGINX y Telegraf) utilizó menos del 40 % de CPU y alrededor de 3,5 GB de memoria, sin variaciones bruscas asociadas a picos de tráfico. La latencia observada entre la generación de un evento de consenso y su disponibilidad en los paneles de Grafana se mantuvo por debajo del segundo en las muestras analizadas, lo que satisface el requisito de monitoreo casi en tiempo real para fenómenos como cambios de *head*, finalización de *epochs* o aparición de blobs.

Cuadro 3.1: Resumen de escenarios de prueba sobre el sistema de monitoreo.

Escenario	Objetivo	Métricas observadas	Resultado
Operación continua 24 h en main-net	Verificar continuidad del flujo de datos en condiciones normales.	7 200 transiciones de <i>head</i> y 225 de <i>finalized_checkpoint</i> esperadas para el periodo; ausencia de huecos en las series de bloques y eventos.	Sin pérdida de eventos en los flujos monitorizados; series completas para bloques <i>head/finalized</i> .
Latencia del evento al dashboard	Confirmar monitoreo casi en tiempo real.	Diferencia entre marca temporal del evento de consenso y su aparición en Grafana: mediana ~ 350 ms, percentil 95 % < 750 ms.	Objetivo cumplido; las variaciones de <i>head</i> , <i>epochs</i> y <i>blobs</i> se visualizan en < 1 s.
Consumo de recursos del stack	Evaluar viabilidad operativa del despliegue.	Uso conjunto de CPU inferior al 40 % y memoria en torno a 3,5 GB (NiFi ~ 2 GB, InfluxDB ~ 600 MB, Grafana ~ 250 MB); ausencia de <i>swap</i> o saturación de disco durante la campaña.	El stack opera con holgura en el host de pruebas, dejando margen para extender métricas y dashboards.
Resiliencia ante cortes de la Beacon API	Comprobar la capacidad de recuperación frente a fallos transitorios de red.	Simulación de interrupciones breves en una de las fuentes de la Beacon API (ventanas de ~ 5 min); observación de reinicios y reconexión del cliente SSE; comparación de series entre ambas fuentes.	La fuente restante mantiene las series de bloques <i>head/finalized</i> sin huecos; tras la reconexión no se detectan duplicaciones en las métricas agregadas.
Reinicio controlado de servicios internos	Validar la recuperación del pipeline ante reinicios de InfluxDB y Grafana.	Reinicio secuencial de InfluxDB y Grafana con NiFi en ejecución; análisis del vaciado de colas internas y del tiempo de recuperación de dashboards.	Las colas de NiFi absorben el retraso y se vacían en menos de 60 s tras la reactivación de InfluxDB; los paneles de Grafana se resincronizan sin pérdida de datos en el intervalo experimental.

En cuanto al comportamiento de la red, durante las 24 horas consideradas se registraron las 7 200 transiciones de *head* y las 225 actualizaciones de *finalized_checkpoint* esperadas para Ethereum en Prueba de Participación, sin pérdidas de eventos en los flujos monitorizados. Los dashboards de estado del nodo confirmaron que el cliente Beacon permaneció sincronizado en todo momento, con distancias de sincronización cercanas a cero y sin episodios de desconexión prolongada de la capa de ejecución. No se observaron reorganizaciones de cadena ni eventos de *slashing* en la ventana analizada, lo que se refleja en series temporales planas para esos indicadores, mientras que las métricas de blobs mostraron valores compatibles con la adopción esperada de EIP-4844 (hasta seis blobs por bloque en los períodos de mayor actividad).

Estos resultados permiten concluir que el sistema de monitoreo propuesto es capaz de sostener la cadencia natural de la Beacon Chain post-*Merge*, ingerir y correlacionar métricas de consenso y ejecución desde fuentes redundantes, y exponerlas en dashboards reproducibles con un coste de recursos moderado. Además, la ausencia de duplicados en las series de bloques *head/finalized* respalda la robustez del pipeline frente a fallos transitorios de red o de servicios externos, cumpliendo así con los objetivos de fiabilidad y observabilidad definidos para el trabajo.

CAPÍTULO 4

Herramientas desarrolladas

4.1. Librería Java para Escucha de Eventos de la Beacon Chain

4.1.1. Propósito

Como parte del presente trabajo se desarrolló una librería Java de propósito específico para la escucha en tiempo real de eventos emitidos por la Beacon Chain de Ethereum, consumiendo el canal Server-Sent Events (SSE) del endpoint estándar `/eth/v1/events` provisto por la Beacon API. La librería mantiene una conexión persistente, permite suscripciones selectivas a tópicos, implementa reconexión automática y expone un modelo de callbacks/listeners seguro para hilos, con el fin de desacoplar la adquisición del stream de su posterior procesamiento.

Este componente constituye la fuente de datos en tiempo real del stack de monitoreo propuesto, alimentando el pipeline de NiFi → InfluxDB → Grafana para observabilidad, analítica y alertas. El diseño prioriza ligereza, robustez, flexibilidad y facilidad de uso.

4.1.2. Comparativa, estado del arte y motivación del desarrollo propio

La alternativa más directa en java para esta finalidad es la librería Web3j.

En particular, web3j (core) —la biblioteca Java más difundida para el execution layer— ofrece tres tipos de filtros: bloques, transacciones pendientes y filtros por topics (logs de contratos). Esto es adecuado para eventos EVM tradicionales, pero no cubre por sí sola el modelo de Server-Sent Events (SSE) del Beacon Node API (consensus layer), que es el estándar actual para escuchar eventos de la Beacon Chain (por ejemplo, head, finalized_checkpoint, attestation,

`proposer_slashing`, `attester_slashing`, `blob_sidecar`, `block_gossip`, entre otros). La documentación oficial de web3j detalla que soporta esos tres filtros de ejecución, sin mencionar SSE ni endpoints de beacon [41], lo que evidencia el alcance centrado en ejecución y no en consenso.

Para salvar esa brecha, Web3 Labs publicó en 2020 un cliente Beacon Node API para web3j (web3j-eth2), que efectivamente implementa suscripción a eventos SSE del endpoint `/eth/v1/events` del estándar de Eth2 y, en su primera versión, escuchaba los topics `head`, `block`, `attestation`, `voluntary_exit`, `finalized_checkpoint` y `chain_reorg`. El anuncio y ejemplo de uso muestran explícitamente la suscripción simultánea a esos temas vía SSE [42]. Sin embargo, ese cliente quedó anclado a la especificación v0.12.2 (release 1.0.0, enero de 2021) y no registra nuevas releases desde entonces [43], [44]. Desde 2023–2024 el Beacon API incorporó nuevos eventos (p. ej., `blob_sidecar` con EIP-4844/Deneb; `proposer_slashing` y `attester_slashing` como eventos; y `block_gossip`), de acuerdo con las release notes oficiales del repositorio `ethereum/beacon-APIs`. Esto implica que bibliotecas no mantenidas activamente podrían no cubrir el conjunto actual de eventos del consenso.

En el plano del execution layer, web3j (core) arrastra incidencias documentadas en la decodificación de parámetros no indexados y arreglos dinámicos de eventos (por ejemplo, `DynamicArray` y `string[]`), que afectan la fiabilidad al reconstruir event payloads. Estos problemas están reportados en issues públicos de web3j y workarounds conocidos por la comunidad [45], [46], [47], lo que refuerza la necesidad de un pipeline de eventos sobrio y verificable cuando el requisito es auditoría de consenso de alto volumen.

El estándar del Beacon Node API define la suscripción SSE mediante `/eth/v1/events ?topics=...` y proveedores/documentación de clientes de consenso (Teku [48], Validations Cloud [49], QuickNode [19], etc.) recogen este mecanismo y la semántica de los eventos (incluyendo campos como `slot`, `block`, `state`, `epoch transition`, etc.). Este patrón SSE es el fundamento de la estrategia de streaming continuo de la Beacon Chain que la librería desarrollada implementa y robustece (reintentos, backoff, tolerancia a cortes, parseo tipado y soporte de nuevos topics).

Cuadro 4.1: Cuadro comparativo de listeners y alcance de eventos en Java para Ethereum.

Aspecto	web3j (core, ejecución)	web3j-eth2 (Beacon Node API, 2020/21)	Eth-events-listener (este desarrollo, MIT)
Capa/protocolo	<i>Execution layer</i> vía JSON-RPC (<code>eth.*</code>). Filtros: bloques, pendientes, <i>topics/logs</i> . No expone SSE [41].	<i>Consensus layer</i> vía REST + SSE (<code>/eth/v1/events</code>). Diseñado para Beacon Chain [42].	<i>Consensus layer</i> vía REST + SSE con soporte extendido y actualizado a <i>topics</i> recientes [8].
Modo de suscripción	<i>Polling</i> con filtros (<code>eth_newFilter</code> , <code>eth_getFilterChanges</code>). Propenso a “Filter not found” por caducidad/tiempos [41, 50].	SSE nativo (<i>EventSource</i>). El ejemplo oficial suscribe <code>head</code> , <code>block</code> , <code>attestation</code> , <code>voluntary_exit</code> , <code>finalized_checkpoint</code> , <code>chain_reorg</code> [42].	SSE nativo con <i>reconnect</i> exponencial, <i>heartbeats</i> , reanudación idempotente y <i>parsers</i> tipados [8].
Cobertura de eventos Beacon	N/A (no Beacon; centrado en eventos EVM del <i>execution layer</i>) [41].	Cubre <i>topics</i> iniciales (spec v0.12.2, 2020). Sin nuevas <i>releases</i> desde 2021 [42, 43].	Cubre <i>topics</i> vigentes (incluyendo <code>proposer_slashing</code> , <code>attester_slashing</code> , <code>blob_sidecar</code> , <code>block_gossip</code> , etc.) según <i>release notes</i> de <i>beacon-APIs</i> [44, 51, 8]. Si: por ejemplo, manejo de <code>blob_sidecar</code> acorde a las extensiones de Beacon API para EIP-4844/Deneb [20, 52, 44].
Eventos Deneb/EIP-4844	No aplica (no Beacon, sin soporte para <code>blob_sidecar</code>).	No documentado en la versión 1.0.0 del cliente Eth2 [43].	Igual formato Beacon API, con mapeos a modelos internos, validaciones y <i>schema</i> versionado orientado a ingestión y monitoreo [51, 8]. Decodificación explícita por tipo y validación de <i>schema</i> por versión de API, priorizando trazabilidad y auditoría [8].
Formato de datos	<i>Logs</i> EVM con <i>topics</i> y <i>data</i> en hexadecimal. Decodificación ABI en el cliente [41, 53].	JSON SSE conforme Beacon API; <i>payload</i> con campos como <code>slot</code> , <code>block</code> , <code>state</code> y <code>flags</code> [19, 42].	
Robustez de decodificación	Incidencias conocidas con parámetros no indexados y arreglos dinámicos; existen <i>workarounds</i> en <i>issues</i> públicos [45, 46, 47].	Sin incidencias públicas relevantes, pero alcance limitado al conjunto de eventos de 2020 [42].	
Estado de mantenimiento	Activo, licencia Apache-2.0 [54].	Última <i>release</i> 1.0.0 (enero 2021), sin actualizaciones posteriores [43].	Activo en este proyecto (2025), publicado bajo licencia MIT y diseñado para reutilización por terceros [8].
Casos de uso típicos	DApps Java, <i>backends</i> EVM y <i>indexers</i> de logs/contratos [41, 54].	<i>Proof-of-concepts</i> y utilidades básicas de Beacon en el contexto 2020–2021 [42].	Monitoreo de consenso, métricas, auditoría y <i>data ingestion</i> de la Beacon Chain, integrable con pipelines como NiFi → InfluxDB → Grafana [8].

4.1.3. Objetivos y requisitos

Objetivo general

Proveer una librería abierta, eficiente y reutilizable para la captura de eventos de la Beacon Chain con énfasis en facilidad de integración y robustez operativa.

Objetivos específicos

- Cubrir los principales tópicos SSE de la Beacon API (p. ej., `attestation`, `attester_slashing`, `blob_sidecar`, `block`, `bls_to_execution_change`, `chain_reorg`, `contribution`

```
_and_proof, finalized_checkpoint, head, light_client_finality_update, light_client  
_optimistic_update, proposer_slashing, voluntary_exit).
```

- Exponer una API para suscripciones múltiples y entrega no bloqueante de mensajes.
- Incorporar reconexión automática ante caídas, timeouts o reinicios del nodo.
- Ofrecer un modelo de datos genérico (JSON) apto para flujos ETL y control de cardinalidad.
- Publicar el artefacto en Maven Central bajo licencia MIT, fomentando su adopción.

Requisitos no funcionales (NFR):

- Rendimiento: latencia de entrega baja y sobrecarga mínima de CPU/memoria.
- Confiabilidad: reconexión con backoff y preservación de suscripciones.
- Seguridad: compatibilidad con HTTPS/TLS y manejo seguro de endpoints y tokens.
- Portabilidad: compatibilidad con Java 21 y ejecutable en entornos Linux/Windows.

4.1.4. Alcance

La librería aborda exclusivamente la ingestión de eventos vía SSE de la Beacon Chain. El procesamiento, la persistencia y la visualización se delegan a los componentes del proyecto donde se esté utilizando.

4.1.5. Arquitectura y diseño

Patrones aplicados

Se adopta el patrón Observador para la distribución de eventos, Builder y Factory para la configuración/creación del cliente e Inversión de Control para la inyección de listeners. La estructura de paquetes separa el cliente SSE, los modelos y la configuración.

Componentes principales

- Capa de cliente: `BeaconEventClient`, `OkHttpBeaconEventClient`, `BeaconEventClientBuilder`, `BeaconEventClientFactory`.
- Capa de modelo: `BeaconEvent` (payload como JSON) y `EventType` (enumeración de tópicos).
- Resiliencia: reconexión automática (reintentos con backoff), gestión thread-safe de listeners y control de ciclo de vida (start/stop/close).

Flujo de ejecución (alto nivel)

:

1. Construcción del cliente (endpoint Beacon, red, tópicos, timeouts).
2. Apertura de la conexión SSE y registro de listeners por tópico.
3. Recepción de eventos, parseo JSON y publicación a listeners.
4. Enrutamiento del consumidor (NiFi u otro) hacia persistencia/alertas.
5. Reintentos automáticos ante fallas manteniendo suscripciones activas.

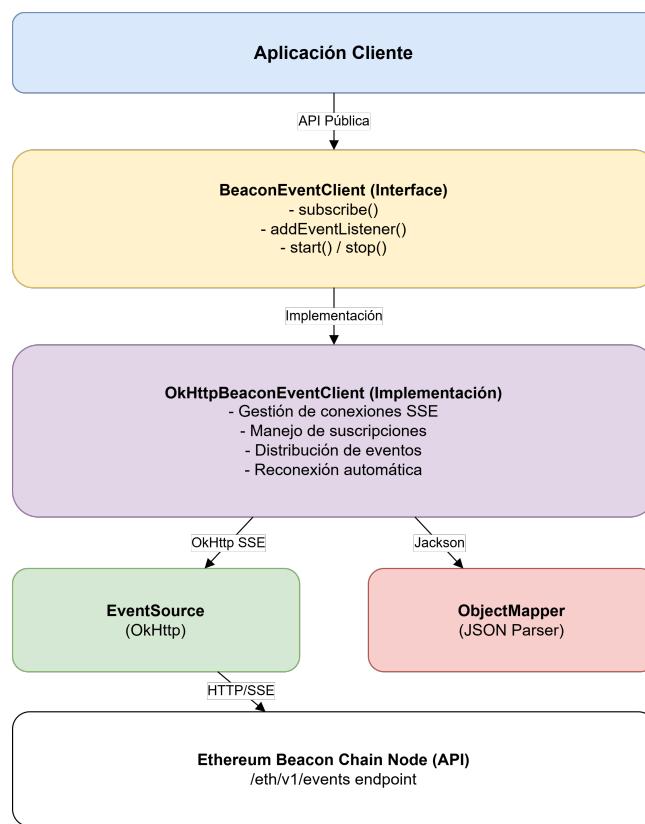


Figura 4.1: Arquitectura de la librería Java para la escucha de eventos de la Beacon Chain: la aplicación cliente interactúa con la interfaz **BeaconEventClient**, cuya implementación **OkHttpBeaconEventClient** gestiona las conexiones SSE, las suscripciones, la distribución de eventos y la reconexión automática utilizando **EventSource (OkHttp)** y **ObjectMapper (Jackson)** para consumir el endpoint `/eth/v1/events` del nodo de Ethereum Beacon Chain.

4.1.6. API pública y uso

Coordenadas Maven (publicación en Maven Central)

```
<dependency>
<groupId>io.github.am-ilcar</groupId>
```

```

<artifactId>eth-events-listener</artifactId>
<version>1.0.2</version>
</dependency>

```

Ejemplo de uso (suscripción a múltiples tópicos)

Algoritmo 4.1 Suscripción a eventos de la Beacon Chain

Require: Endpoint de la Beacon API, conjunto de tópicos

Ensure: Eventos recibidos y entregados a los consumidores registrados

Crear cliente de eventos con:

endpoint, tópicos, *connectTimeoutSeconds* y *readTimeoutSeconds*

Registrar listener para el tópico HEAD que:

recibe cada evento HEAD y lo envía downstream (ej.: a NiFi vía HTTP/Queue)

Registrar listener para el tópico BLOCK que:

actualiza métricas (altura de bloque, *timestamp*, etc.)

Iniciar el cliente para abrir la conexión SSE a la Beacon API

while la aplicación está en ejecución **do**

 recibir eventos desde la Beacon API

 despachar cada evento al listener correspondiente según su tópico

Configuración típica

- endpoint: URL del Beacon node (mainnet/testnets).
- topics: conjunto de tópicos a escuchar, por ejemplo:
attestation, attester_slashing, blob_sidecar, block,
bls_to_execution_change, chain_reorg, contribution_and_proof,
finalized_checkpoint, head, light_client_finality_update,
light_client_optimistic_update, proposer_slashing, voluntary_exit.
- connectTimeoutSeconds / readTimeoutSeconds: control de timeouts.
- maxRetries / backoffStrategy: política de reconexión (según implementación).
- addListener(EventType, Consumer<BeaconEvent>): registro de consumidores.

4.1.7. Integración con el stack

La librería es la fuente primaria de eventos en tiempo real. Se ha implementado en el desarrollo de un procesador y controlador de NiFi que se detallará más adelante.

El consumo en este trabajo es: Beacon SSE → (Listener) → Controlador NiFi → Procesador NiFi → InfluxDB → Grafana.

Lineamientos de integración

- NiFi: modelar un Controlador y procesador de entrada que reciba el payload JSON por tópico emitido por esta librería y lo aplane (flatten) a campos atómicos; agregar tags como network, source, topic.

4.1.8. Robustez, rendimiento y operación

En términos prácticos, la librería se diseñó para poder ejecutarse de forma continua sin intervención manual, incluso cuando se producen fallas de red o variaciones en la carga de trabajo. A continuación se describen los aspectos más relevantes de su comportamiento en un entorno de operación real.

Cuando la conexión SSE se interrumpe, el cliente intenta restablecerla de manera automática. El tiempo de espera entre reintentos aumenta progresivamente hasta un límite configurable, de modo a evitar saturar al nodo Beacon y reducir el riesgo de bloqueos. La entrega de eventos a los distintos listeners se realiza en hilos separados del hilo de recepción, pero con un número máximo de hilos acotado; de esta forma se aprovecha el paralelismo sin generar excesiva contención ni competencia por recursos compartidos.

La librería expone contadores básicos, como la cantidad de eventos procesados, el tiempo entre eventos consecutivos y el número de reintentos de conexión. Estos indicadores pueden integrarse con un sistema de métricas externo y sirven para diagnosticar problemas de rendimiento o disponibilidad. El flujo completo de conexión, recepción y procesamiento está instrumentado con mensajes de log en distintos niveles (por ejemplo, INFO, DEBUG y WARN), lo que permite separar el registro de operación normal de los detalles de depuración y de las condiciones anómalas.

Finalmente, el diseño busca mantener un consumo moderado de memoria y CPU. Para ello, los eventos se manejan inicialmente como cadenas JSON sin decodificarlos de inmediato, y solo se transforman cuando es necesario, lo que facilita además que el *payload* se entregue directamente a NiFi para su procesamiento posterior.

4.1.9. Seguridad y cumplimiento

En materia de seguridad, la comunicación con el nodo Beacon se realiza sobre HTTPS/TLS, verificando los certificados del lado del cliente para evitar ataques de intermediario. La superficie de ataque se mantiene reducida: la aplicación no expone puertos adicionales, sino que es el cliente saliente el que abre y mantiene la conexión SSE hacia el nodo. En cuanto al cumplimiento legal y la reutilización, la librería se publica bajo licencia MIT, lo que habilita su uso comercial, modificación y redistribución, con la condición de conservar el aviso y el texto completo de la licencia.

4.1.10. Validación y pruebas

La validación del comportamiento se apoya en un conjunto de pruebas unitarias que cubren el parseo del JSON recibido, el enrutamiento correcto de los eventos hacia los listeners registrados y las transiciones del ciclo de vida (`start/stop/close`). Adicionalmente, se llevaron a cabo pruebas de resiliencia mediante la inyección controlada de fallas (cortes de red y *timeouts*), verificando que la librería sea capaz de reconectarse sin producir duplicación de callbacks ni perder eventos. Por último, se comprobó la compatibilidad de ejecución sobre Java 21 en los sistemas operativos Linux y Windows, lo que cubre los entornos de despliegue previstos en este trabajo.

4.1.11. Publicación, versionado y mantenimiento

El código fuente se aloja en un repositorio público bajo el nombre `eth-events-listener`, lo que permite auditar la implementación y recibir contribuciones de terceros. El artefacto compilado se publica en Maven Central con las coordenadas indicadas en este capítulo, de manera que pueda integrarse fácilmente en proyectos Java que utilicen un gestor de dependencias estándar. Para el versionado se adopta el esquema SemVer (`x.y.z`) y se mantiene un *changelog* donde se documentan cambios relevantes, en particular aquellos que introducen rupturas de compatibilidad. El canal de soporte y mejora continua se organiza a través de *issues* y *pull requests*, que sirven tanto para reportar fallas como para proponer nuevas funcionalidades.

4.1.12. Limitaciones y trabajo futuro

Una limitación relevante es la cobertura de tipos de eventos soportados. La evolución del protocolo Ethereum puede introducir nuevas estructuras o campos que aún no están contemplados en la librería, por lo que será necesario ampliarla a medida que se publiquen nuevas versiones de la especificación de la Beacon Chain y aparezcan casos de uso que requieran esos eventos adicionales.

4.1.13. Impacto académico y replicabilidad

La librería materializa un aporte verificable y reutilizable: está publicada bajo licencia MIT y en Maven Central, lo que facilita su replicación por terceros, su evaluación independiente y su incorporación en otros proyectos académicos e industriales que requieran ingestión fiable de eventos de la Beacon Chain.

4.2. Controlador y procesador open-source para NiFi con el fin de recibir eventos de un nodo

4.2.1. Propósito

El nifi-eth-events-bundle tiene por propósito integrar en tiempo real los eventos de la Ethereum Beacon Chain (vía SSE en `/eth/v1/events`) al stack de observabilidad del proyecto (NiFi → InfluxDB → Grafana), con bajo acoplamiento, resiliencia y output estándar en JSON o InfluxDB Line Protocol (LP).

Concretamente, el bundle:

- expone un *Controller Service* que gestiona una conexión SSE única y compartida hacia el beacon node (creación perezosa, reconexión, salud),
- y un Processor que suscribe eventos, convierte los payloads a JSON/LP y emite FlowFiles listos para persistencia y visualización.

4.2.2. Motivación del desarrollo propio

El desarrollo de un bundle propio para NiFi se justifica, en primer lugar, por una brecha funcional clara en la herramienta. No existía un procesador nativo que se suscriba a *server-sent events* (SSE) de la Beacon Chain ni que convierta esos eventos al formato Line Protocol con tipado y *tags* consistentes. Los procesadores genéricos como `InvokeHTTP` o `Fetch` no resuelven adecuadamente conexiones SSE de larga duración ni gestionan el *backpressure* típico de los flujos continuos.

Además, tras *The Merge* y actualizaciones como Deneb/EIP-4844, la Beacon API amplió en forma significativa su conjunto de *topics* (por ejemplo, `head`, `finalized_checkpoint`, `chain_reorg`, `blob_sidecar`, entre otros). Se requería una pieza capaz de seguir el estándar SSE vigente y de incorporar nuevos *topics* a medida que aparecen, sin romper los flujos de NiFi ya configurados ni obligar a redefinir la lógica de ruteo.

Por otro lado, el trabajo exige series temporales confiables tanto para análisis como para visualización. Para lograrlo, el Processor implementa mapeos a Line Protocol deterministas (enteros con sufijo `i`, valores `float`, booleanos y cadenas escapadas) y define *tags* como `network`, `source`, `event_type` y `endpoint`, lo que permite construir consultas Flux más limpias y reproducibles sobre InfluxDB.

Finalmente, se buscó independencia tecnológica y facilidad de mantenimiento. Algunas bibliotecas o *wrappers* públicos (por ejemplo, clientes `eth2` antiguos o soluciones alternativas basadas en `web3j`) no cubren el conjunto actual de eventos ni contemplan un patrón SSE con reconexión robusta. Optar por un desarrollo propio, publicado bajo licencia MIT, ofrece margen para evolucionar la cobertura de eventos (nuevos *topics*), controlar el rendimiento y mantener una documentación alineada con el sistema implementado en este trabajo.

4.2.3. Objetivos y requisitos

Objetivo general

Proveer un bundle de NiFi que integre eventos de consenso de Ethereum (SSE) a pipelines ETL, con foco en cobertura de tópicos, latencia baja, resiliencia, y compatibilidad con bases de series temporales (InfluxDB).

Objetivos específicos

- Implementar un Controller Service que gestione la conexión SSE.
- Implementar un Processor que consuma el stream y emita JSON o Line Protocol, con suscripción selectiva a eventos.
- Exponer propiedades configurables claras (service, event types, formato de salida, red para LP).
- Incorporar transformaciones LP por tipo de evento (p. ej., attestation, blob_sidecar) y atributos para ruteo en NiFi.

Requisitos no funcionales

- Resiliencia: reconexión automática y cancelación ordenada en NiFi.
- Rendimiento: buffering y procesamiento en `onTrigger()` con colas internas; emisión en REL_SUCCESS.
- Compatibilidad: NiFi 2.x; uso de Java 21 consistente con el sistema.

4.2.4. Alcance

El bundle cubre la ingestión de eventos de la capa de consenso vía SSE y su transformación a formatos aptos para series temporales. El almacenamiento y la visualización se delegan en InfluxDB y Grafana, ya presentes en el sistema.

4.2.5. Arquitectura y diseño

Módulos Maven

El proyecto se organiza en seis módulos (API/impl/NAR para servicio y procesador), siguiendo las convenciones de NiFi y separando contratos, implementaciones y empaquetado.

- nifi-eth-events-bundle (pom)
- nifi-eth-events-services-api (+nar)
- nifi-eth-events-services (+nar)

- nifi-eth-events-processors (+nar)

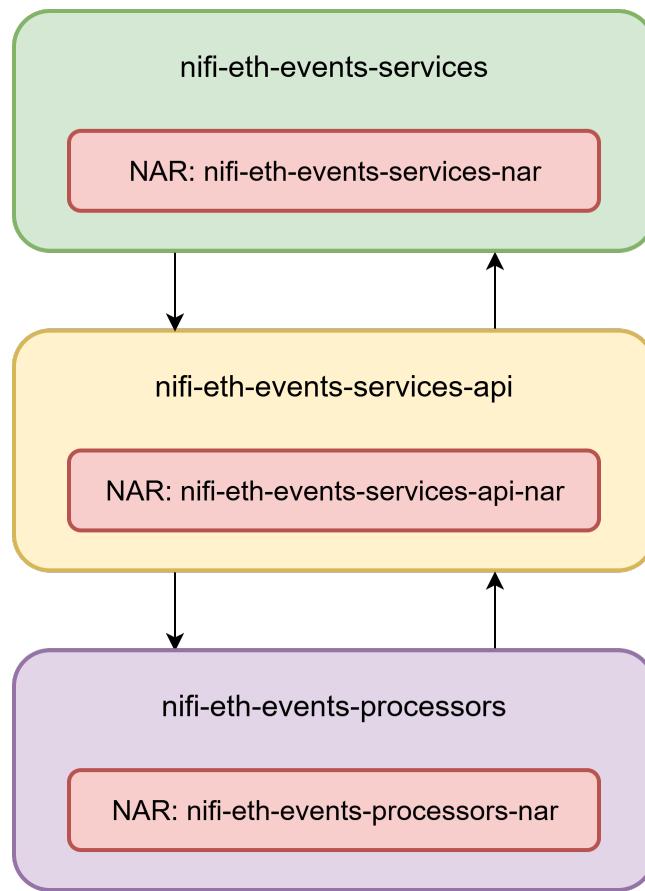


Figura 4.2: Componentes del proyecto `nifi-eth-events`: módulo de servicios, API y procesadores, cada uno empaquetado como un NAR independiente.

Componentes principales

- BeaconEventsService (API): contrato para suscribir/desuscribir event types, obtener la URL del nodo y chequear salud. Modelo con `EventType` y `BeaconEvent`.
- StandardBeaconEventsService (Controller Service): crea una conexión SSE compartida con inicialización perezosa, reconexión y agregación dinámica de topics; maneja errores con counters y rate-limit de logs.
- BeaconEventsProcessor (Processor): consume eventos, crea FlowFiles y escribe JSON o Line Protocol; agrega atributos (tipo, timestamp, URL) y enruta a SUCCESS.

4.2.6. API pública y uso (configuración en NiFi)

Processor – BeaconEventsProcessor

- Ethereum Events Service (ControllerService, requerido)
- Tipos de eventos (lista separada por comas, validada contra `EventType`)

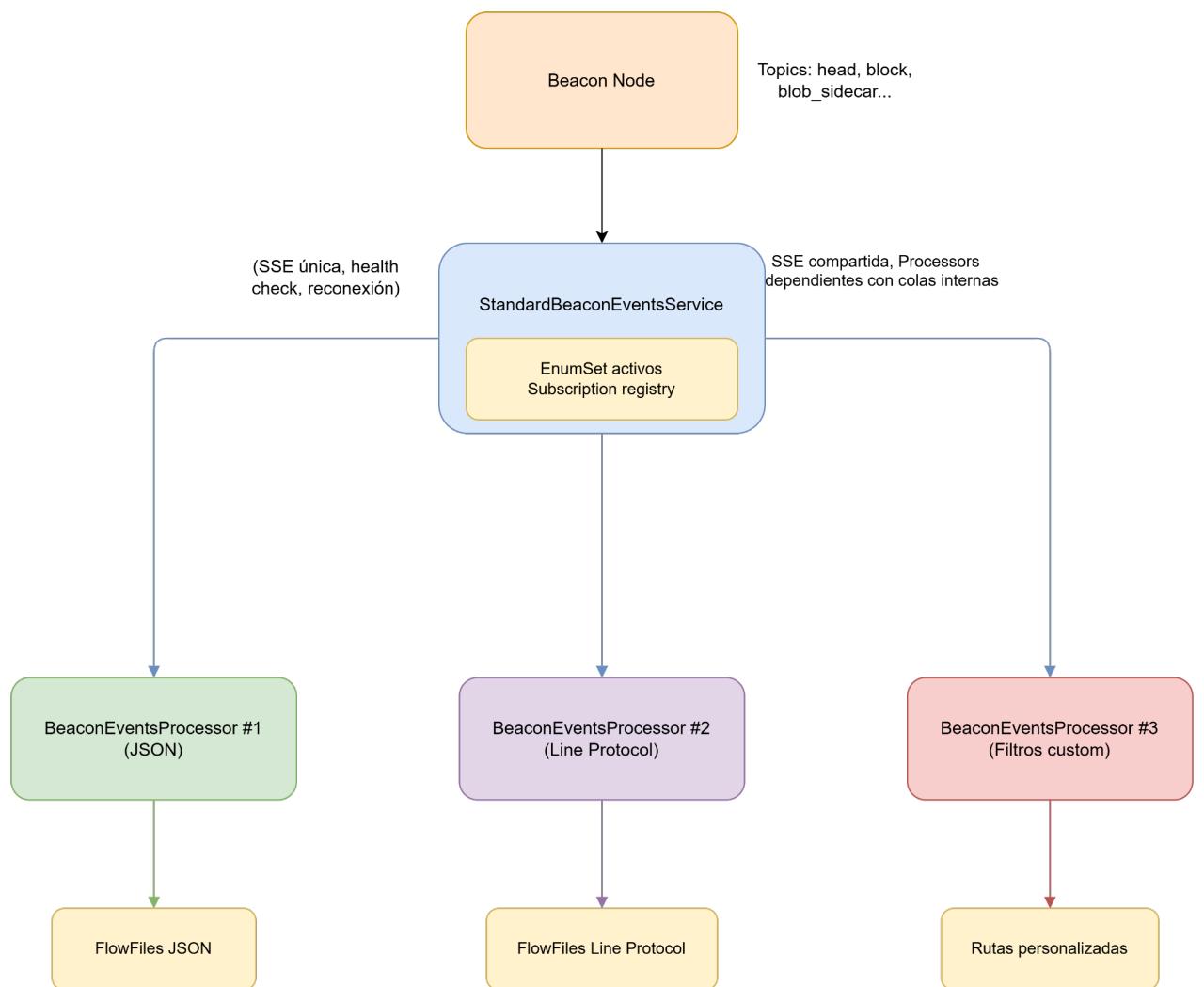


Figura 4.3: Diagrama de componentes y flujo interno: el servicio `StandardBeaconEventsService` concentra la suscripción SSE al `Beacon Node` y distribuye los eventos a múltiples `BeaconEventsProcessor`, que generan `FlowFiles` en formato JSON, Line Protocol o rutas personalizadas según la configuración.

- Formato de salida = JSON — LINE_PROTOCOL (default: JSON)
- Network (opcional; visible solo si salida es Line Protocol; ej. *mainnet*, *holesky*)

Controller Service – StandardBeaconEventsService

- Conexión SSE mediante la librería de listener (creación perezosa, reconexión, normalización de URL).

Ciclo de operación

- `onScheduled()` suscribe topics y registra callback.
- `onTrigger()` drena cola, crea FlowFiles, escribe JSON o compone LP y enruta a SUCCESS.
- `onStopped()` cancela suscripciones.

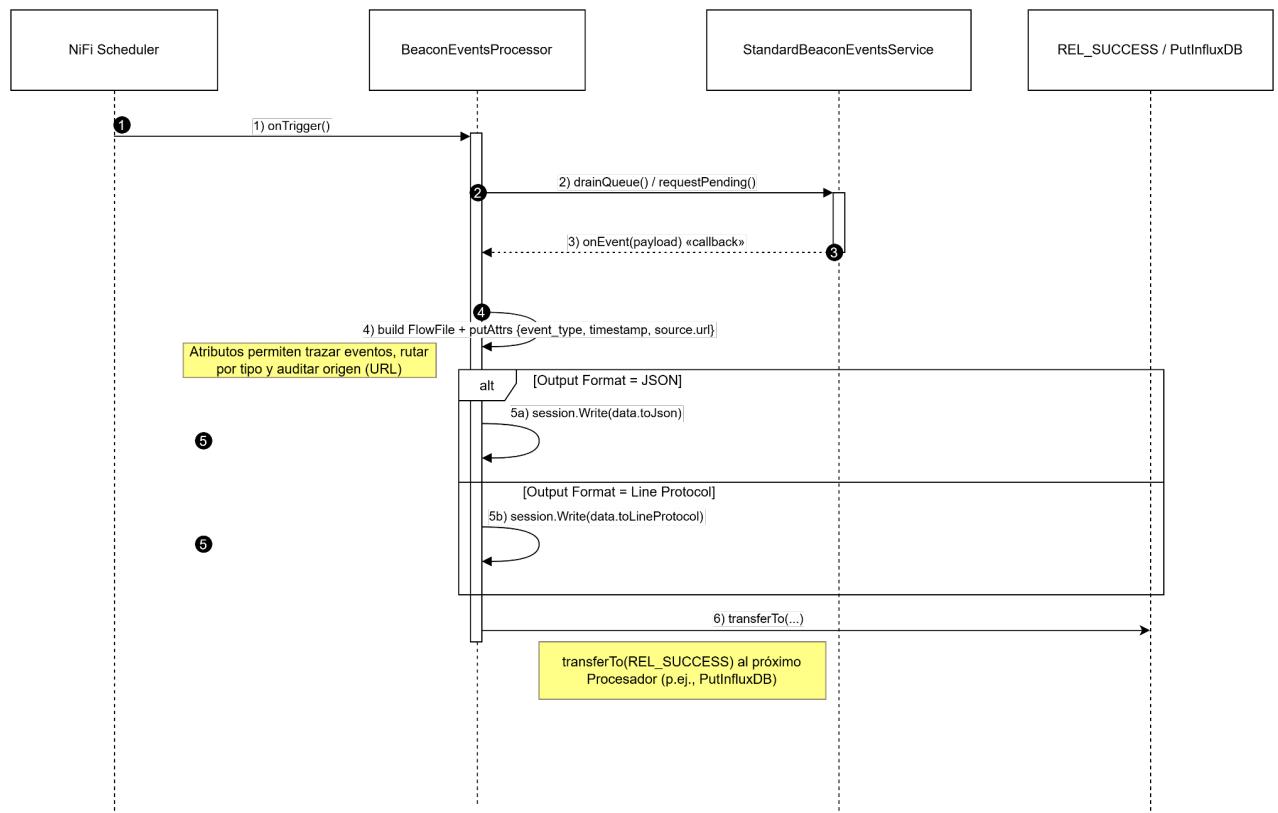


Figura 4.4: Diagrama de secuencia del ciclo `onTrigger()` en NiFi: el *NiFi Scheduler* dispara el **BeaconEventsProcessor**, que consulta al **StandardBeaconEventsService**, construye el *FlowFile* con atributos de tipo de evento, marca de tiempo y URL de origen, decide entre serializar en JSON o Line Protocol y finalmente transfiere el resultado por la relación `REL_SUCCESS` al siguiente procesador.

4.2.7. Integración con el stack

El *bundle* se integra en la cadena de monitoreo existente basada en eventos SSE del *Beacon Node*. El servicio de NiFi recibe los eventos SSE, el procesador los transforma al formato requerido y luego los envía a la base de datos de series temporales, desde donde son visualizados en los tableros de Grafana. En términos generales, el flujo queda: Beacon SSE → (servicio y procesador en NiFi) → InfluxDB → Grafana.

InfluxDB almacena las mediciones con etiquetas (*tags*) que identifican la red, el origen y el tipo de evento o punto de acceso (*endpoint/event_type*), y con campos (*fields*) en formato Line Protocol. De esta manera se reutilizan los tableros existentes en Grafana sin necesidad de redefinir consultas ni paneles.

Un flujo típico de uso en NiFi es el siguiente: el servicio `StandardBeaconEventsService` se encuentra habilitado, el `BeaconEventsProcessor` está configurado para suscribirse a los tipos de evento `head`, `block` y `finalized_checkpoint`, producir salida en `LINE_PROTOCOL` y apuntar a la red `mainnet`. La relación de éxito (`success`) del procesador se conecta a un `PutInfluxDB`, que escribe las mediciones y habilita su visualización inmediata en los tableros de Grafana existentes.

4.2.8. Transformación a InfluxDB Line Protocol

El *processor* convierte payloads JSON de eventos en LP con tipado (enteros con i, cadenas escapadas, booleans, floats) y tags estandarizadas. Ejemplo ATTESTATION (simplificado) y esquema LP:

```
measurement,tag1=v1,tag2=v2 field1=v,field2=v timestamp
```

La salida LP es ingerida por InfluxDB, donde ya se adoptaron measurements como `beacon_event`, `beacon_block_info`, `beacon_node_info`, etc., con consultas Flux en Grafana.

4.2.9. Robustez, rendimiento y operación

La arquitectura del bundle está pensada para minimizar el impacto sobre el nodo Beacon y, al mismo tiempo, ofrecer un comportamiento predecible en producción. La conexión SSE se mantiene única y compartida en el *Controller Service*, que se encarga de reconectarse ante fallos y de gestionar las altas y bajas de suscripciones activas. De este modo se evita abrir múltiples conexiones redundantes y se reduce el consumo de recursos del nodo.

Por su parte, el *Processor* realiza el procesamiento de eventos de forma concurrente, apoyado en una cola interna que desacopla la llegada de datos de su emisión hacia el flujo de NiFi. Cada llamada a `onTrigger()` controla cuántos eventos se consumen de la cola y construye los *FlowFiles* correspondientes, añadiendo atributos específicos que facilitan el ruteo posterior y la depuración (por ejemplo, tipo de evento, marca de tiempo u origen).

Finalmente, el manejo de errores incluye contadores internos y un límite de frecuencia para los mensajes de log, lo que permite detectar problemas recurrentes sin inundar los registros.

Esta combinación de conexión única, cola de procesamiento y control de logging aporta robustez y hace que el comportamiento operativo del bundle sea estable incluso bajo alta carga o ante fallos intermitentes de la red.

4.2.10. Seguridad y cumplimiento

El diseño del bundle contempla explícitamente los aspectos de seguridad en la comunicación con el nodo Beacon. Todas las conexiones se realizan solamente hacia el punto de acceso configurado, y la URL se normaliza y valida antes de establecer la suscripción a los eventos. Esto reduce el riesgo de errores de configuración y evita que se utilicen destinos no previstos.

En cuanto a los datos tratados, tanto el formato Line Protocol como el JSON generado por el procesador contienen únicamente información pública expuesta por la *Beacon Chain*, sin incluir credenciales ni secretos de configuración. De este modo, los *FlowFiles* pueden ser almacenados y reenviados dentro del flujo de NiFi sin introducir requisitos adicionales de protección de datos sensibles.

Finalmente, el bundle y la biblioteca subyacente se distribuyen bajo licencia abierta MIT, lo que facilita su uso, modificación y redistribución por parte de terceros. Esta elección favorece la auditoría independiente del código y el alineamiento con buenas prácticas de transparencia y cumplimiento en proyectos de software libre.

4.2.11. Validación y pruebas

La implementación del bundle se acompañó de un conjunto de pruebas automatizadas basadas en `nifi-mock` y `JUnit`, lo que permitió validar su comportamiento sin depender de un flujo completo en ejecución. Estas pruebas cubren la configuración de propiedades del procesador y del servicio, verificando que los valores obligatorios sean correctamente interpretados y que las combinaciones inválidas se rechacen de forma explícita.

Asimismo, se ejercitan escenarios de suscripción múltiple a distintos tipos de evento, comprobando que el *Controller Service* registre y gestione adecuadamente cada suscriptor. Se valida también la generación de salida en formato JSON, asegurando que la estructura del *FlowFile* y sus atributos coincidan con lo esperado para cada variante.

Finalmente, se incluyen casos de prueba orientados al ciclo de vida de los componentes (métodos `onEnabled`, `onDisabled` y `onStopped`), así como al manejo de errores de conexión y a la lógica de reintentos. De esta manera se verifica que el bundle pueda recuperarse de fallos transitorios sin comprometer la estabilidad del flujo de NiFi.

Es importante destacar que la versión pública del bundle no incluye la opción de generar *FlowFiles* en formato Line Protocol. Para los fines específicos de este proyecto se utilizó una variante de Line Protocol que no cumple de manera estricta con la estructura oficial y completa del formato ya que ciertos tipos de eventos de la *Beacon Chain* obligan a apartarse de dicha estructura para poder representarlos adecuadamente. Para evitar posibles confusiones o usos

incorrectos en otros contextos, esta modalidad de salida se mantuvo fuera de la distribución pública.

4.2.12. Publicación, versionado y mantenimiento

El proyecto se distribuye open-source (MIT) con artefactos NAR y coordenadas Maven públicas; se mantienen releases y documentación técnica, con enfoque en compatibilidad NiFi 2.x.

4.2.13. Limitaciones y trabajo futuro

El diseño del bundle está condicionado por la evolución de la Beacon API. Cambios en los esquemas de datos o la aparición de nuevos tipos de eventos pueden requerir ajustes en el mapeo actual, tanto en la lógica del servicio como en la transformación aplicada por el procesador. Esta dependencia implica que, a medida que la especificación avance, será necesario revisar periódicamente las suscripciones y las estructuras de salida para mantener la compatibilidad.

Por otra parte, el trabajo realizado se centra en un conjunto acotado de formatos de salida y en un flujo de eventos de volumen moderado. Como línea de mejora se plantea incorporar formatos adicionales de serialización, como Avro, Parquet o Protobuf, que resultan más adecuados para escenarios de integración con lagos de datos o sistemas de análisis masivo. En el mismo sentido, sería conveniente introducir mecanismos de agrupamiento configurable (*batching*) que permitan enviar varios eventos en una sola operación, optimizando así el rendimiento en entornos de alto volumen.

4.2.14. Impacto académico y replicabilidad

El bundle materializa una integración replicable entre consenso de Ethereum y pipelines de datos: su licenciamiento MIT y empaquetado NAR facilitan la evaluación independiente, adopción por terceros y extensión hacia nuevas métricas y tableros de observabilidad.

CAPÍTULO 5

CONCLUSIÓN

En este trabajo se abordó el problema de la observabilidad en Ethereum posterior a *The Merge*, en un contexto en el que la plataforma combina un mecanismo de consenso de Prueba de Participación con una arquitectura de doble capa que incrementa la complejidad operativa [1, 2, 3]. El punto de partida fue la constatación de que los tableros de monitoreo tradicionales, centrados exclusivamente en métricas de infraestructura o de la máquina virtual de ejecución, resultan insuficientes para explicar fenómenos propios del consenso moderno, tales como episodios de degradación de la finality o variaciones abruptas en la participación de validadores [6]. A partir de este diagnóstico, el objetivo general fue diseñar y materializar un sistema de monitoreo para Ethereum post-Merge que fuese reproducible, abierto y sustentable, integrando tanto un marco conceptual como artefactos de software específicos. Los objetivos específicos planteados en la Introducción —marco teórico, arquitectura de referencia, librería para eventos de consenso, bundle de NiFi y construcción de dashboards— orientaron el desarrollo de los capítulos posteriores.

En relación con el marco teórico, el Capítulo 2 sistematizó los conceptos fundamentales necesarios para interpretar métricas de la red y de los nodos. Se revisaron los principios de blockchain, la arquitectura post-Merge y la separación entre capa de consenso y capa de ejecución, así como la noción de tiempo en la Beacon Chain mediante slots y epochs [3, 12]. También se describieron la estructura mínima de los bloques PoS y su *execution payload*, el modelo de gas después de EIP-1559 y la incorporación de blobs con EIP-4844, junto con el funcionamiento operativo del staking y los validadores. Finalmente, se analizó en detalle la Beacon Node API y sus eventos SSE, destacando aquellos que resultan críticos para el monitoreo (por ejemplo, `head`, `finalized_checkpoint`, `blob_sidecar`, `chain_reorg`), así como parámetros de salud del nodo y de la red. Este recorrido permitió construir un mapa conceptual que vincula directamente las métricas visibles en un dashboard con hechos del protocolo.

Sobre esa base, el Capítulo 3 presentó el sistema bajo estudio y la arquitectura general del entorno propuesta. Se diseñó un stack modular compuesto por Apache NiFi, InfluxDB y Grafana, desplegado mediante Docker Compose para garantizar la reproducibilidad y la portabilidad del entorno [37, 38, 39]. NiFi se encargó de la ingestión, transformación y enrutamiento de los datos; InfluxDB actuó como repositorio de series temporales para eventos y métricas; y Grafana proporcionó la capa de visualización y alertas. La descripción detallada de la arquitectura, de las características del entorno y de las distintas fuentes de datos (incluyendo endpoints REST y canales SSE de la Beacon API) mostró cómo se articula un pipeline de datos orientado específicamente al monitoreo de Ethereum, y no solo a la telemetría de infraestructura genérica. La contenedorización mediante Docker Compose y el uso de componentes ampliamente adoptados en la industria contribuyeron a que el entorno resulte replicable en laboratorios y entornos de ensayo, y permitieron instrumentar una campaña experimental de 24 horas sobre mainnet que sirvió para cuantificar volúmenes de datos, latencias extremo a extremo y consumo de recursos del stack.

El Capítulo 4 describió los aportes tecnológicos desarrollados en el marco de este trabajo. En primer lugar, se implementó y publicó la librería Java `eth-events-listener`, capaz de mantener suscripciones SSE al endpoint `/eth/v1/events`, gestionar reconexiones y exponer un modelo de callbacks tipado que facilita su integración con otros sistemas [8]. En segundo lugar, se diseñó y publicó el bundle `nifi-eth-events-bundle`, compuesto por un Controller Service y un Processor para Apache NiFi, que permiten recibir el stream de eventos de consenso, aplanar su estructura y emitirlos en JSON o InfluxDB Line Protocol hacia el motor de series temporales [9]. Ambos componentes se liberaron bajo una licencia permisiva y se empaquetaron como artefactos reutilizables, contribuyendo al ecosistema open source y habilitando la replicabilidad de los resultados.

La validación del sistema combinó pruebas automatizadas sobre la librería y el bundle —incluyendo inyección controlada de fallas de red y verificación de reconexiones— con ejecuciones del entorno de monitoreo desplegado, desde la recepción de eventos de la Beacon Chain hasta su representación en paneles de Grafana. En particular, la campaña de 24 horas descrita en el Capítulo 3 y resumida en la Tabla 3.1 permitió observar que el stack sostiene sin pérdidas la cadencia natural de bloques *head* y *finalized*, mantiene latencias de ingestión y visualización por debajo del segundo y opera con un consumo moderado de CPU y memoria. Se constató así que la arquitectura propuesta permite observar en tiempo casi real indicadores tales como la evolución de la ranura actual, la participación de validadores, la frecuencia de eventos de reorganización, la relación entre bloques *head* y *finalized*, y métricas de gas y blobs asociadas a la capa de ejecución. Estos resultados, coherentes con la metodología basada en revisión documental, diseño iterativo de arquitectura y validación experimental, indican que al combinar conceptos teóricos con un pipeline de datos diseñado específicamente para este contexto es posible reducir la brecha entre la especificación formal del protocolo y la práctica cotidiana de operación y monitoreo.

En cuanto al cumplimiento de los objetivos específicos planteados en la Introducción, el

trabajo entrega evidencias claras para cada uno de ellos. El primer objetivo, relativo a la elaboración de un marco conceptual actualizado, se satisface con el desarrollo del Capítulo 2; el segundo, orientado al diseño de una arquitectura de referencia de ingesta, transformación, almacenamiento y visualización, se materializa en el Capítulo 3 y se refuerza con los resultados experimentales obtenidos sobre mainnet; el tercero y el cuarto, referidos a la librería de eventos SSE y al bundle para NiFi, se concretan en los componentes open source publicados y en su integración efectiva con el stack de monitoreo; finalmente, el quinto objetivo, centrado en la construcción de dashboards y reglas de alerta, se aborda a través de los paneles de Grafana configurados y de su capacidad para apoyar tanto la operación cotidiana como el análisis de incidencias. En conjunto, estos logros permiten afirmar que el objetivo general de diseñar y materializar un sistema de monitoreo para Ethereum post-Merge que sea reproducible, abierto y sustentable se ha cumplido.

Las principales contribuciones de este Trabajo Final de Grado son: la elaboración de un marco teórico actualizado y orientado explícitamente al monitoreo de Ethereum post-Merge, el diseño de una arquitectura de observabilidad basada en componentes open source de amplio uso en la industria de ingeniería de datos, el desarrollo y publicación de una librería Java específica para eventos de la Beacon Chain y de un bundle de NiFi que integra dichos eventos a pipelines ETL y la demostración, en un entorno controlado y reproducible, de que este stack permite construir paneles y alertas que reflejan con fidelidad el estado del consenso y de los nodos. Estas contribuciones combinan rigor académico y aplicabilidad práctica, en línea con los objetivos formativos de la carrera de Ingeniería en Informática y con el énfasis del trabajo en replicabilidad y software abierto.

No obstante, el trabajo presenta limitaciones. La validación se centró en un conjunto acotado de redes y clientes de consenso, por lo que sería conveniente ampliar las pruebas a otros entornos, incluyendo redes de prueba y mainnet en distintos proveedores de infraestructura. Asimismo, el modelo actual de dashboards se concentra en un conjunto representativo de métricas, pero no agota todas las posibilidades que ofrece la Beacon API ni las señales de la capa de ejecución. Finalmente, si bien el bundle y la librería fueron diseñados para ser extensibles, su evolución futura dependerá de la incorporación de nuevas características del protocolo (por ejemplo, cambios en la especificación de eventos o nuevas EIPs relevantes para el consenso).

Como líneas de trabajo futuro se propone: extender la cobertura de eventos y métricas monitoreadas, incorporando indicadores adicionales de rendimiento y seguridad; integrar el stack con sistemas de orquestación como Kubernetes para evaluar su comportamiento en despliegues de mayor escala; explorar técnicas de detección automática de anomalías sobre las series temporales recolectadas; y estudiar la aplicabilidad del enfoque a otros protocolos de Prueba de Participación. De este modo, la propuesta presentada puede evolucionar desde un caso de estudio centrado en Ethereum hacia un marco general para la observabilidad de sistemas distribuidos basados en blockchain.

REFERENCIAS

- [1] *The Merge*. ethereum.org. URL: <https://ethereum.org/roadmap/merge/> (visitado 22-10-2025).
- [2] *Proof-of-Stake (PoS)*. ethereum.org. URL: <https://ethereum.org/developers/docs/consensus-mechanisms/pos/> (visitado 14-11-2025).
- [3] *Beacon Chain - Ethereum Consensus Specs*. URL: <https://ethereum.github.io/consensus-specs/specs/phase0/beacon-chain/> (visitado 14-11-2025).
- [4] Barnabé Monnot. *Visualising the 7-Block Reorg on the Ethereum Beacon Chain*. The price of agency. 29 de mayo de 2022. URL: <https://barnabe.substack.com/p/pos-ethereum-reorg> (visitado 19-11-2025).
- [5] *Slashing on Ethereum: Everything You Need to Know*. RockX. 13 de feb. de 2023. URL: <https://blog.rockx.com/ethereum-slashing-101/> (visitado 14-11-2025).
- [6] Offchain Labs. *Post-Mortem Report: Ethereum Mainnet Finality (05/11/2023)*. Off-chain Labs. 23 de ago. de 2023. URL: <https://medium.com/offchainlabs/post-mortem-report-ethereum-mainnet-finality-05-11-2023-95e271dfd8b2> (visitado 15-11-2025).
- [7] *Beacon-API*. URL: <https://ethereum.github.io/beacon-APIs/#/> (visitado 22-10-2025).
- [8] Am-ilcar. *Am-Ilcar/Eth-Events-Listener*. 30 de jun. de 2025. URL: <https://github.com/Am-ilcar/eth-events-listener> (visitado 18-11-2025).
- [9] *Maven Central: Io.Github.Am-Ilcar:Nifi-Eth-Events-Bundle*. Maven Central. URL: <https://central.sonatype.com/artifact/io.github.am-ilcar/nifi-eth-events-bundle> (visitado 18-11-2025).
- [10] *Nodes and Clients*. ethereum.org. URL: <https://ethereum.org/en/developers/docs/nodes-and-clients/> (visitado 21-09-2024).
- [11] Ethereum Improvement Proposals. *EIP-1559: Fee Market Change for ETH 1.0 Chain*. Ethereum Improvement Proposals. URL: <https://eips.ethereum.org/EIPS/eip-1559> (visitado 14-11-2025).
- [12] *Prueba de participación (PoS)*. ethereum.org. URL: <https://ethereum.org/es/developers/docs/consensus-mechanisms/pos/> (visitado 14-11-2025).

- [13] *Bloques*. ethereum.org. URL: <https://ethereum.org/es/developers/docs/blocks/> (visitado 14-11-2025).
- [14] *Guía de Ethereum*. ethereum.org. URL: <https://ethereum.org/es/whitepaper/> (visitado 14-11-2025).
- [15] Cong Nguyen, Hoang Dinh Thai, Diep Nguyen, Dusit Niyato, Huynh Nguyen y Eryk Dutkiewicz. «Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities». En: *IEEE Access* PP (26 de jun. de 2019), págs. 1-1. DOI: 10.1109/ACCESS.2019.2925010.
- [16] *Attestations*. ethereum.org. URL: <https://ethereum.org/developers/docs/consensus-mechanisms/pos/attestations/> (visitado 14-11-2025).
- [17] *Networking Layer*. ethereum.org. URL: <https://ethereum.org/developers/docs/networking-layer/> (visitado 14-11-2025).
- [18] Kerala Blockchain Academy. *Ethereum 2.0: The Beacon Chain*. KBA Ethereum Stories. 22 de jul. de 2022. URL: <https://medium.com/ethereum-stories/ethereum-2-0-the-beacon-chain-d669fa65e50d> (visitado 18-11-2025).
- [19] */Eth/v1/Events RPC Method — Ethereum Docs*. URL: <https://www.quicknode.com/docs/ethereum/eth-v1-events> (visitado 18-10-2025).
- [20] *Arquitectura del nodo*. ethereum.org. URL: <https://ethereum.org/es/developers/docs/nodes-and-clients/node-architecture/> (visitado 18-11-2025).
- [21] 0xBreadguy. *WTF Is Finality?* 0xBreadguy's Substack. 11 de oct. de 2024. URL: <https://0xbreadguy.substack.com/p/wtf-is-finality> (visitado 14-11-2025).
- [22] *Ethereum Gas and Fees: Technical Overview*. ethereum.org. URL: <https://ethereum.org/developers/docs/gas/> (visitado 14-11-2025).
- [23] ¿Qué es EIP 1559? URL: <https://academy.bit2me.com/que-es-eip-1559/> (visitado 18-11-2025).
- [24] Ethereum Improvement Proposals. *EIP-4844: Shard Blob Transactions*. Ethereum Improvement Proposals. URL: <https://eips.ethereum.org/EIPS/eip-4844> (visitado 14-11-2025).
- [25] *Danksharding*. ethereum.org. URL: <https://ethereum.org/ka/roadmap/danksharding/> (visitado 14-11-2025).
- [26] *Explicación del EIP-4844: Blobs, tasas y más*. URL: <https://www.datawallet.com/es/cripto/eip-4844-explained> (visitado 18-11-2025).
- [27] *Impact Of EIP-4844 On Ethereum: What You Need To Know*. Hacken. URL: <https://hacken.io/discover/eip-4844-explained/> (visitado 18-11-2025).

- [28] Ethereum Improvement Proposals. *EIP-4895: Beacon Chain Push Withdrawals as Operations*. Ethereum Improvement Proposals. URL: <https://eips.ethereum.org/EIPS/eip-4895> (visitado 14-11-2025).
- [29] *Slashing on Ethereum: Everything You Need to Know - RockX*. URL: <https://blog.rockx.com/ethereum-slashing-101/> (visitado 18-11-2025).
- [30] *Ethereum*. URL: <https://aandds.com/blog/ethereum.html> (visitado 18-11-2025).
- [31] *Post-Shanghai: What Really Happened*. Glassnode Insights - On-Chain Market Intelligence. 8 de mayo de 2023. URL: <https://insights.glassnode.com/the-week-onchain-week-19-2023/> (visitado 15-11-2025).
- [32] Metrika. *Shapella Recap*. Medium. 20 de abr. de 2023. URL: <https://blog.metrika.co/shapella-recap-139856de0533> (visitado 19-11-2025).
- [33] */Eth/v1/Node/Syncing RPC Method — Ethereum Docs*. URL: <https://www.quicknode.com/docs/ethereum/eth-v1-node-syncing> (visitado 18-11-2025).
- [34] */Eth/v1/Node/Peer_count RPC Method — Ethereum Docs*. URL: https://www.quicknode.com/docs/ethereum/eth-v1-node-peer_count (visitado 18-11-2025).
- [35] *Infura - Home — Web3 Development Platform — IPFS API & Gateway — Blockchain Node Service*. Infura. URL: <https://www.infura.io> (visitado 19-11-2025).
- [36] *Alchemy - the Web3 Development Platform*. Alchemy. URL: <https://www.alchemy.com/> (visitado 19-11-2025).
- [37] *Apache NiFi*. Apache NiFi. URL: <https://nifi.apache.org/> (visitado 18-11-2025).
- [38] *InfluxDB — Real-time Insights at Any Scale*. InfluxData. Sat, 15 Jan 2022 15:32:09 +0000. URL: <https://www.influxdata.com/home/> (visitado 18-11-2025).
- [39] *Grafana: The Open and Composable Observability Platform*. Grafana Labs. URL: <https://grafana.com/> (visitado 18-11-2025).
- [40] *Nginx - Official Image — Docker Hub*. URL: https://hub.docker.com/_/nginx (visitado 18-11-2025).
- [41] *Filters and Events - Web3j*. URL: https://docs.web3j.io/4.11.0/advanced/filters_and_events/?utm_source=chatgpt.com (visitado 18-10-2025).
- [42] *Announcing Web3j Eth2 Beacon Node API Client*. Web3 Labs Blog. 15 de dic. de 2020. URL: <https://blog.web3labs.com/announcing-web3j-eth2-beacon-node-api-client/> (visitado 18-10-2025).
- [43] *Web3j/Web3j-Eth2*. web3j, 28 de oct. de 2024. URL: <https://github.com/web3j/web3j-eth2> (visitado 18-10-2025).
- [44] *Releases · Ethereum/Beacon-APIs*. GitHub. URL: <https://github.com/ethereum/beacon-APIs/releases> (visitado 18-10-2025).

- [45] *Non Indexed Parameters (DynamicArray) Not Returning Value Correctly* · Issue #1733 · LFDT-web3j/Web3j. GitHub. URL: <https://github.com/LFDT-web3j/web3j/issues/1733> (visitado 18-10-2025).
- [46] *FunctionReturnDecoder Returns Incorrect Result When Decoding ‘string[]’* · Issue #1251 · LFDT-web3j/Web3j. GitHub. URL: <https://github.com/LFDT-web3j/web3j/issues/1251> (visitado 18-10-2025).
- [47] *Wrong Data from Dynamic Array* · Issue #950 · LFDT-web3j/Web3j. GitHub. URL: <https://github.com/LFDT-web3j/web3j/issues/950> (visitado 18-10-2025).
- [48] *ConsenSys Teku REST API Documentation - V23.3.1.* URL: <https://consensys.github.io/teku/?version=v23.3.1> (visitado 18-10-2025).
- [49] */Eth/v1/Events — Validation Cloud.* URL: <https://docs.validationcloud.io/v1/ethereum/beacon-api/eth-v1-events> (visitado 18-10-2025).
- [50] *Understanding Ethereum’s “Filter Not Found.^{Erorr} and How to Fix It.* Chainstack. URL: <https://docs.chainstack.com/docs/understanding-ethereums-filter-not-found-error-and-how-to-fix-it> (visitado 14-11-2025).
- [51] *Ethereum/Beacon-APIs.* ethereum, 13 de nov. de 2025. URL: <https://github.com/ethereum/beacon-APIs> (visitado 14-11-2025).
- [52] *EIP-4844: Shard Blob Transactions.* URL: <https://www.eip4844.com> (visitado 14-11-2025).
- [53] *API JSON-RPC.* ethereum.org. URL: <https://ethereum.org/es/developers/docs/apis/json-rpc/> (visitado 14-11-2025).
- [54] *Maven Central: Org.Web3j:Core.* Maven Central. URL: <https://central.sonatype.com/artifact/org.web3j/core> (visitado 14-11-2025).