Opciones para Servidor Java con Soporte HTTP/3 en Backend

La llegada de **HTTP/3** (basado en **QUIC**, sobre UDP) presenta desafíos para las aplicaciones Java del lado servidor. A continuación se investiga el estado actual de las opciones disponibles para que un backend Java soporte HTTP/3, incluyendo servidores Java que lo implementan directamente, proyectos experimentales con bindings nativos, y soluciones indirectas usando proxies (como Nginx o Caddy) que terminan HTTP/3 frente al cliente. Se detalla para cada opción su estabilidad (experimental o estable), madurez, documentación y adecuación para entornos de producción.

Servidores Java con soporte HTTP/3 directo

Estos servidores y frameworks Java pueden manejar conexiones HTTP/3 de forma nativa (es decir, el propio servidor Java implementa QUIC/HTTP3 sin necesidad de un proxy externo). En general, el soporte nativo en Java todavía es **experimental** en la mayoría de casos, debido a la complejidad de QUIC (requiere handshake TLS 1.3 integrado, IO sobre UDP, etc.). Muchos proyectos recurren a bibliotecas nativas (vía JNI/JNA) para la capa QUIC. A continuación, se revisan los principales servidores:

Jetty (Eclipse Jetty)

El servidor Jetty ha sido pionero en adoptar HTTP/3 en Java. Jetty 10.0.8/11.0.8 introdujo un módulo opcional para HTTP/3 en 2022 [1] [2]. Jetty implementa QUIC/HTTP3 **utilizando la biblioteca nativa Quiche** de Cloudflare, a través de JNA (Java Native Access) o las APIs de memoria nativa de Java 17 (Proyecto Panama), ya que no era factible implementar QUIC en Java puro [3].

Estabilidad: El soporte HTTP/3 en Jetty se considera **experimental** y **no recomendado para entornos productivos** por ahora [3]. El objetivo inicial fue lograr una implementación funcional más que optimizada, dado que al momento de su lanzamiento la especificación IETF de HTTP/3 aún estaba en borrador [4]. En 2024/2025, Jetty 12 continúa ofreciendo este soporte como módulo opcional. No hay indicaciones públicas de que se haya marcado como "estable" aún, por lo que se debe usar con precaución.

Documentación: Jetty proporciona tanto APIs de alto nivel (p.ej. `HTTP3ServerConnector` para embebido) como un módulo para habilitar HTTP/3 en el servidor standalone [2]. La documentación oficial describe cómo activar el conector HTTP/3 y señala las limitaciones actuales. En resumen, Jetty ofrece un camino experimental para usar HTTP/3 en Java si se está dispuesto a lidiar con dependencias nativas y posibles problemas de rendimiento.

Netty y frameworks basados en Netty (Reactor Netty, Quarkus, Micronaut, Vert.x)

Netty es un framework de bajo nivel ampliamente usado en servidores embebidos Java. Actualmente, **Netty no incluye HTTP/3 en su versión estable** (4.1.x), pero tiene un módulo **incubador experimental** para QUIC/HTTP3. Este se compone de `netty-incubator-codec-quic` (implementa QUIC apoyándose

internamente en *quiche*) y `netty-incubator-codec-http3` (implementa la capa HTTP/3 sobre QUIC) [5] [6]. Estas librerías proporcionan el soporte necesario siempre que se incluyan como dependencias y se carguen las librerías nativas correspondientes (por ejemplo, Netty utiliza quiche vía JNI; de hecho, su codec QUIC utiliza quiche según CVE-2025-29908 [6]).

Varios frameworks populares en Java construidos sobre Netty han habilitado este soporte experimental recientemente:

- **Reactor Netty (Spring Boot WebFlux):** La versión Reactor Netty 1.2 (Reactor 2024.0, incluido en Spring Boot 3.4) añadió soporte **experimental** para HTTP/3 [7]. Para activarlo, se debe incluir la dependencia incubadora (`io.netty.incubator:netty-incubator-codec-http3`) y configurarlo en el servidor embebido de Spring Boot [8] [9]. Esto permite que aplicaciones Spring WebFlux y Spring Cloud Gateway hablen HTTP/3, aunque bajo riesgo experimental [10].

- **Micronaut:** A partir de Micronaut 4 (lanzado en 2023), el framework incluye soporte experimental para HTTP/3 utilizando el proyecto incubador de Netty [11]. Micronaut 4 actualiza su servidor HTTP (basado en Netty) para permitir HTTP/3 y también integra *io_uring* experimentalmente para mejorar IO [11]. Al igual que en Spring, requiere las dependencias Netty adecuadas. Sigue siendo marcado como experimental por la propia documentación de Micronaut [11].

- **Quarkus:** En Quarkus 3 (2023), se anunció soporte para HTTP/3 aprovechando las novedades de la pila reactiva subyacente. Quarkus 3 usa **Vert.x** 4 (que a su vez usa Netty) y comenzó a integrar Netty 5 (aún en desarrollo) para obtener HTTP/3 "out of the box" [12]. En otras palabras, Quarkus habilita QUIC/HTTP3 a nivel de servidor reactivo de forma transparente. Sin embargo, dado que esto depende de Netty en fase incubadora/preview, se considera **experimental**. No hay evidencias de que Quarkus 3.x recomiende HTTP/3 para producción; más bien es una capacidad presente para experimentar.

En todos estos casos basados en Netty, la **estabilidad es limitada**. El soporte es **experimental**, con posibles problemas de rendimiento o compatibilidad. Por ejemplo, Reactor Netty lo denomina "experimental support" explícitamente [10], y Micronaut también [11]. No se recomienda aún para producción a gran escala hasta que Netty estabilice el codec HTTP/3 (posiblemente en Netty 4.2 o 5.x futuros [13]). La documentación de cada framework provee guías para habilitarlo pero con advertencias.

Undertow (WildFly / JBoss)

Undertow, el servidor web embebido de JBoss/WildFly, *no soporta actualmente HTTP/3 de forma nativa*. Undertow históricamente se basaba en la librería IO *XNIO*, y en la serie 2.x soportaba HTTP/1.1 y HTTP/2 pero no QUIC. En 2019 se anunció Undertow 3.0 con un cambio importante: reemplazar XNIO por Netty como motor subyacente [14]. Esto abría la puerta a nuevas funcionalidades de transporte en Undertow (por ejemplo, aprovechar HTTP/2 de Netty, WebSockets, etc.). De hecho, en ese anuncio se mencionó que Undertow 2.x probablemente **no recibiría soporte HTTP/3** debido a su antiguo stack, delegando esas mejoras a la rama 3.x en adelante [15].

A pesar de ello, **a fecha de 2025 Undertow no anuncia soporte QUIC/HTTP3 estable**. Es posible que alguna versión 3.x experimental pudiera integrarlo (heredando la capacidad de Netty incubator), pero **no hay documentación oficial** ni releases con HTTP/3. Los usuarios de Undertow/WildFly que requieran HTTP/

3 típicamente deben situar un proxy front-end (NGINX, etc.) que termine QUIC y reenvíe a Undertow via HTTP/1.1 o 2. En resumen, Undertow hoy **no es adecuado** para HTTP/3 directo (quizá en el futuro, si Netty 4.2+/5 se integra plenamente).

Apache Tomcat

Tomcat es otro servidor Java muy popular (implementación de referencia de Servlet/JSP). **Actualmente Tomcat *no* soporta HTTP/3 de forma nativa**. Hubo intenciones iniciales de implementarlo (especialmente de cara a Tomcat 11, que es la versión alineada con Jakarta EE 11), pero se decidió **postergar/descartar el soporte HTTP/3** en Tomcat por el momento [16]. Principalmente esto se debe a consideraciones de rendimiento y arquitectura: QUIC implica manejar TLS 1.3 y UDP en el propio servidor, lo cual es intensivo en CPU; el equipo de Tomcat sugiere que, en escenarios de producción, es preferible delegar TLS/QUIC a un proxy dedicado (como Nginx) para que Tomcat siga enfocado en la lógica de aplicación [16]. De hecho, Tomcat considera que la mejor práctica es usar un proxy externo para terminación TLS (y ahora QUIC) en otra máquina, en lugar de integrar QUIC en el conector interno [17].

Estado: Las versiones actuales (Tomcat 10.1, 11.0) **no tienen soporte**. Si en el futuro se llega a implementar HTTP/3 en Tomcat, probablemente utilizará APIs nativas vía Project Panama y una pila QUIC nativa en C/C++ [18]. Por ahora, Tomcat no ofrece nada experimental siquiera. Para **entornos productivos con Tomcat**, la **solución es obligatoriamente usar un proxy** (ver más adelante) que maneje HTTP/3 con el cliente y hable HTTP/1.1 o 2 con Tomcat. Esto fue confirmado en la documentación de Tomcat 11: los planes para QUIC se "shelvearon" por falta de demanda y complejidad [16].

Proyectos experimentales de HTTP/3 en Java (JNI, implementaciones puras)

Existen proyectos comunitarios que no son servidores web completos por sí mismos, sino **bibliotecas o stacks experimentales** para QUIC/HTTP3 en Java. Suelen usarse para investigación, pruebas de concepto o para integrar en aplicaciones especializadas. Entre ellos destacan:

Quiche4j (JNI a *quiche*)

Quiche4j es una librería que expone en Java la implementación de QUIC/HTTP3 de Cloudflare (*quiche*, escrita en Rust). Provee una API de bajo nivel para manejar paquetes QUIC y estado de conexiones, haciendo llamadas nativas via JNI al código Rust [19]. En esencia, Quiche4j es un *binding* del cliente/servidor QUIC de quiche para poder usarlo desde la JVM. La aplicación debe encargarse de la parte de *network I/O* (sockets UDP) y los timers, mientras que Quiche4j se encarga de la lógica de protocolo (frames QUIC, criptografía TLS 1.3, etc.) [20] [21]. Se ha prestado atención al rendimiento, buscando minimizar copias de datos entre la parte nativa y Java y evitando asignaciones innecesarias [22].

Estado: Quiche4j es todavía un proyecto con versiones 0.x (por ejemplo, la versión 0.2.5 es la más reciente en Maven Central). Es mantenido por la comunidad (aprox. 100 estrellas en GitHub). **No está catalogado como estable**; más bien es una herramienta para desarrolladores interesados en QUIC en Java con alto rendimiento. Requiere compilar/usar librerías nativas Rust (cargo) y es multiplaforma (Linux, etc., soportado mediante clasificadores JNI en Maven) [23] [24]. No hay indicios de uso en producción a gran escala. Documentación está disponible en su README y ejemplos (incluye scripts de ejemplo para un servidor

HTTP/3 básico usando la librería [25] [26]). En resumen, **Quiche4j es experimental**; ofrece control granular y acceso completo a QUIC (por ex., permite configurar parámetros de quiche directamente), pero demanda experiencia en JNI y Rust para su correcto uso.

Kwik y Flupke (implementación *pura Java* de QUIC/HTTP3)

Un enfoque alternativo lo ofrece **Kwik**, una implementación de QUIC 100% en Java (escrita por Peter Doornbosch). A diferencia de otros, Kwik no depende de código nativo externo: implementa el protocolo QUIC v1 y v2 en la JVM pura [27], usando las primitivas de TLS 1.3 de Java (JSSE) para cifrado. Encima de *Kwik* se construyó **Flupke**, que es una implementación del protocolo HTTP/3 *sobre* Kwik (añadiendo el manejo de streams, mapeo HTTP, QPACK, etc.) [28]. Juntos, Kwik+Flupke permiten tener un servidor HTTP/3 escrito completamente en Java.

Estado y madurez: A pesar de ser "puros Java", Kwik/Flupke **aún son proyectos en desarrollo activo** y no ampliamente utilizados. Kwik afirma implementar **todas las funcionalidades esenciales de QUIC** (TLS 1.3, 0-RTT, streams bidi/uni, reconexión, actualizaciones de clave, incluso la extensión de datagramas) excepto la migración de conexión en modo servidor, que sigue en trabajo [29]. Se realizan pruebas de interoperabilidad regularmente contra otras implementaciones de QUIC, en las cuales Kwik suele estar **entre los más compatibles** (pasa la mayoría de casos de prueba) [30]. Sin embargo, el propio autor señala que **no está optimizado en rendimiento** todavía y que ciertas condiciones de red extremas no han sido exhaustivamente testeadas [31]. Por tanto, **su idoneidad para producción es relativa**: puede funcionar para casos de uso específicos, pero se provee "as is" sin garantías [31]. Además, la implementación de TLS usada es casera y no auditada por expertos, lo que podría ser un riesgo de seguridad en entornos sensibles [32].

En conclusión, *Kwik/Flupke* demuestran que es posible un servidor HTTP/3 enteramente en Java, pero **por ahora se consideran experimentales**. Pueden ser una opción interesante para experimentar sin dependencias nativas, o para entornos controlados donde se prefiera Java puro a costa de performance. Su documentación está disponible en GitHub/Bitbucket, incluyendo un *README* detallado y guías de uso con Maven [33] [32].

Soluciones con proxies HTTP/3 (indirectas)

Dado que el soporte nativo en servidores Java aún es inmaduro, una estrategia común es usar un **proxy o balanceador de carga** al frente que **hable HTTP/3 con los clientes**, pero luego se comunique con el backend Java vía HTTP/1.1 o HTTP/2. Así se obtiene la mejora de HTTP/3 en la conexión cliente <-> proxy, sin modificar el servidor Java existente. Varias tecnologías populares de proxy ya soportan HTTP/3 (algunas de forma experimental, otras más estable):

- **NGINX:** El servidor Nginx (en su rama mainline 1.25.0+) introdujo soporte para QUIC y HTTP/3 [34]. Para habilitarlo es necesario compilar Nginx con el módulo `--with-http_v3_module` y usar una librería SSL compatible con QUIC (p. ej. BoringSSL, QuicTLS o LibreSSL moderna) [35] [36]. Nginx puede entonces escuchar en puertos UDP QUIC (indicando `listen 443 quic reuseport` en la config) y manejar HTTP/3. **Estabilidad:** Nginx marca este soporte como **"experimental"** oficialmente [34]. Si bien técnicamente funciona (muchos lo usan ya en producción), los desarrolladores advierten *caveat emptor*. A partir de Nginx 1.25 también se incluyó en binarios precompilados para Linux, facilitando pruebas [34]. En producción, Nginx es una opción viable para front-end HTTP/3,

apoyándose en un backend Java tradicional (por ejemplo, terminando QUIC en el puerto 4433 y reenviando internamente a Tomcat en HTTP/1.1 en puerto 8080). Esto añade complejidad (otro servicio más), pero **aísla la lógica QUIC** fuera de la JVM.

- **Caddy:** Caddy Server es un servidor web escrito en Go, conocido por habilitar HTTPS automáticamente. Caddy ha soportado HTTP/3 (mediante la librería quic-go) desde temprano. En su versión 2.6 (2022) se convirtió en **el primer servidor multipropósito en habilitar HTTP/3 por defecto de forma estándar** [37]. Es decir, Caddy desde 2.6 anuncia Alt-Svc y acepta conexiones H3 sin configuración especial. **Estabilidad:** Aunque inicialmente también se etiquetaba como experimental en documentación, el hecho de habilitarlo por defecto indica un alto grado de confianza. En la práctica, Caddy es considerado *production-ready* para HTTP/3, siempre y cuando la pila Go quic-go haya madurado (la cual, tras la estandarización RFC, ha mejorado bastante). Caddy puede actuar como proxy inverso frente a un backend Java: acepta HTTP/3 de los clientes y luego forwardea peticiones HTTP/1.1/2 al puerto del aplicativo Java. Muchos desarrolladores lo usan por su sencillez de configuración y soporte activo.

- **HAProxy:** El proxy de alto rendimiento HAProxy también incorporó soporte HTTP/3/QUIC en versiones recientes (experimento en 2.6, estabilizado en ramas posteriores). Para activarlo se requiere compilar HAProxy con una versión especial de OpenSSL con QUIC (por ejemplo, **QuicTLS** o BoringSSL con parches) [38]. HAProxy 2.8 y Enterprise 3.0 ya traen QUIC habilitado de serie, permitiendo simplemente añadir `bind QUIC` en la configuración para un frontend TLS [38] [39]. **Estabilidad:** HAProxy es reconocido por su robustez; aunque el soporte QUIC era experimental al inicio, a 2024 ya se considera *bastante estable* en su rama principal. Es apto para producción si se configura adecuadamente. HAProxy puede, por ejemplo, terminar H3 en el puerto 443 UDP y mandar a los servidores backend Java vía HTTP/2 sobre TCP. Esta es otra ruta segura mientras los servidores Java nativos no soporten H3.

- **Otros:** *Traefik* (proxy en Go) también soporta HTTP/3 desde versiones 2.9+ de forma similar a Caddy. **Cloudflare** y otros CDNs proporcionan HTTP/3 a los clientes y luego conectan con el origen via protocolos tradicionales, lo cual para algunos escenarios (sitios públicos) es una alternativa (usar un CDN como front). **Apache HTTPD:** El servidor httpd de Apache aún no tiene HTTP/3 en una versión estable oficial; existe un módulo en desarrollo (cortesía de Cloudflare) pero no integrado en builds estándar, por lo que Apache por ahora no es opción a menos que se apliquen parches manualmente.

Conclusión sobre proxies: Usar un proxy HTTP/3 es actualmente la **solución más práctica para entornos de producción** que requieren HTTP/3 con backend Java. Por ejemplo, los propios desarrolladores de Tomcat recomiendan Nginx frente a intentar incorporar QUIC en Tomcat [16]. Estas soluciones indirectas ofrecen *madurez* (particularmente Caddy y HAProxy han sido probados) y permiten aprovechar las ventajas de HTTP/3 (menor latencia en conexiones con el cliente, multiplexación sin bloqueo de cabecera, etc.) sin arriesgar la estabilidad del servidor Java.

Comparativa de opciones

A continuación, se presenta una **tabla comparativa** que resume las distintas opciones discutidas, indicando para cada servidor/proyecto si soporta HTTP/3, de qué tipo es (directo en Java o proxy externo), su estabilidad (experimental vs estable), y notas relevantes:

Servidor/ Proyecto	Soporte HTTP/3	Directo o Proxy	Estabilidad	Notas
Jetty (Eclipse)	Sí (módulo HTTP/3)	Directo (Java)	Experimental	Usa librería nativa Quiche via JNA; *no* recomendado prod. aún [3] [40].
Netty (incubator)	Sí (incubador)	Directo (Java)	Experimental	Codec HTTP/3 sobre QUIC (usa *quiche* por JNI) [6]. Adoptado por Spring (Reactor) [7], Micronaut [11], etc.
Spring Boot/ Reactor	Sí (3.4+ con conf.)	Directo (Java)	Experimental	Requiere Netty incubator HTTP3 e incluir dependencia [8] [10]. Feature *tech preview*.
Quarkus (Vert.x)	Sí (desde 3.x)	Directo (Java)	Experimental	Integra Vert.x/Netty 5 con HTTP/3 out-of-box [12]. Aún no ampliamente probado en prod.
Micronaut	Sí (desde 4.x)	Directo (Java)	Experimental	Soporte HTTP/3 vía Netty incubator anunciado como experimental [11].
Undertow	No (aún)	Directo (Java)	N/A	Sin soporte nativo por ahora. Futuro podría usar Netty (Undertow 3+) pero no implementado [15].
Apache Tomcat	No	Directo (Java)	N/A	Planes cancelados por ahora [16]. Requiere proxy externo (Nginx, Caddy) para HTTP/3.
Quiche4j (JNI)	Sí (librería)	Directo (biblioteca)	Experimental	Binding Java->Rust *quiche*. Bajo nivel, requiere manejo de sockets [19]. Versión 0.x.
Kwik/Flupke (Java)	Sí (biblioteca)	Directo (biblioteca)	Experimental	Implementación pura en Java de QUIC+HTTP3. Alta compatibilidad pero perf. no optimizada [41] [31].
NGINX (1.25+)	Sí (compilar módulo)	Proxy (externo)	Experimental[^1]	Soporte QUIC/H3 desde 1.25 [34]; marcado *experimental*. Usar BoringSSL/QuicTLS [35].
Caddy (2.6+)	Sí (por defecto)	Proxy (externo)	Estable	HTTP/3 habilitado out-of-the-box [37]. Basado en quic-go. Adecuado producción (Go).

Servidor/ Proyecto	Soporte HTTP/3	Directo o Proxy	Estabilidad	Notas
HAProxy (2.6+)	Sí (con BoringSSL)	Proxy (externo)	Estable	Soporta QUIC/H3 (2.6 experimental, >=2.8 estable). Requiere OpenSSL con QUIC [38].

[^1]: “Experimental” en el contexto de Nginx indica que la funcionalidad está en desarrollo; muchas implementaciones lo usan en producción pese a la etiqueta [34].

Referencias: Esta información se recopiló de la documentación oficial y blogs de los proyectos mencionados, incluyendo anuncios de soporte HTTP/3 en frameworks Java (Micronaut [11], Spring Reactor [7]), el blog de Webtide sobre Jetty HTTP/3 [3], el blog de OpenLogic sobre Tomcat 11 [16], y las páginas oficiales de Nginx [34] y otras. Se han indicado las fuentes específicas junto a cada afirmación relevante para mayor detalle. En general, el soporte HTTP/3 en Java backend está emergiendo pero es **mayormente experimental en 2025**, por lo que para entornos críticos se recomienda optar por soluciones de *proxy* hasta que las implementaciones Java maduren. [3] [16]

---

[1] [2] [3] [4] [40] Jetty HTTP/3 Support – Webtide
https://webtide.com/jetty-http-3-support/

[5] GitHub - netty/netty-incubator-codec-http3: Experimental HTTP3 codec on top of QUIC
https://github.com/netty/netty-incubator-codec-http3

[6] CVE-2025-29908 Detail - NVD
https://nvd.nist.gov/vuln/detail/CVE-2025-29908

[7] [8] [9] [10] HTTP/3 support in Reactor 2024.0 Release Train
https://spring.io/blog/2024/11/26/http3-in-reactor-2024/

[11] Micronaut Framework 4.0.0 Released! - Micronaut Framework
https://micronaut.io/2023/07/14/micronaut-framework-4-0-0-released/

[12] Quarkus 3.0.0: A New Era for Java Development | by Samuel Catalano | The Fresh Writes | Medium
https://medium.com/thefreshwrites/quarkus-3-0-0-a-new-era-for-java-development-30d10048d49

[13] HTTP/3 codec support for Netty 4.2 · Issue #14739 - GitHub
https://github.com/netty/netty/issues/14739

[14] [15] Undertow 3.0 Announcement · JBoss Community
https://undertow.io/blog/2019/04/15/Undertow-3.html

[16] [17] [18] Tomcat 11 Overview | OpenLogic
https://www.openlogic.com/blog/tomcat-11-features-preview

[19] [20] [21] [22] [23] [24] [25] [26] GitHub - kachayev/quiche4j: QUIC transport protocol and HTTP/3 for Java
https://github.com/kachayev/quiche4j

[27] [28] [29] [30] [31] [32] [33] [41] GitHub - ptrd/kwik: A QUIC client, client library and server implementation in Java. Supports HTTP3 with "Flupke" add-on.
https://github.com/ptrd/kwik

34 35 36 Support for QUIC and HTTP/3
http://nginx.org/en/docs/quic.html

37 Caddyhttp: Enable HTTP/3 by Default | Hacker News
https://news.ycombinator.com/item?id=32768454

38 39 HTTP | HAProxy config tutorials
https://www.haproxy.com/documentation/haproxy-configuration-tutorials/protocol-support/http/