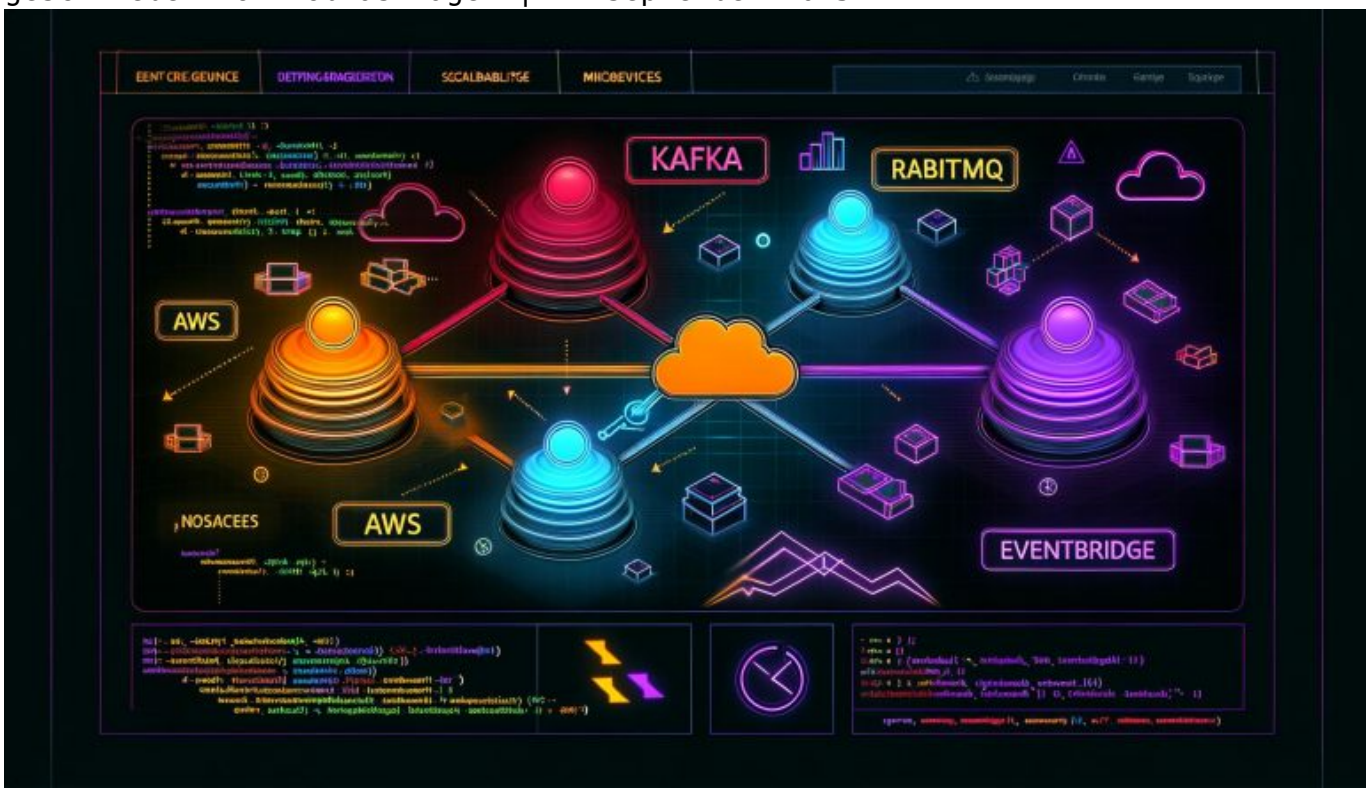
# Event Driven Stack Vergleich: Welcher passt wirklich?

Category: Tools
geschrieben von Tobias Hager | 7. September 2025




# Event Driven Stack Vergleich: Welcher passt wirklich?

Alle reden von „Event Driven“, alle wollen skalieren, alle behaupten, sie hätten den besten Stack. Aber mal ehrlich: Wer checkt wirklich, was hinter Kafka, RabbitMQ, AWS EventBridge und Co. steckt, und welcher Stack für was wirklich taugt? In diesem Artikel gibt’s das schonungslose, technische Deep Dive-Update – ohne Marketing-Blabla, aber mit brutal ehrlicher Antwort, warum dein Event Driven Stack mehr als ein schickes Buzzword sein muss. Spoiler: Wer hier nach Kuschel-Content sucht, ist falsch abgebogen.

- Was ein Event Driven Stack wirklich ist und warum die meisten

Implementierungen daran scheitern
- Die wichtigsten Komponenten: Message Broker, Event Bus, Event Store & Co. – und deren technische Unterschiede
- Vergleich der führenden Technologien: Kafka vs. RabbitMQ vs. AWS EventBridge vs. NATS – mit klaren Pros & Cons
- Worauf es bei der Auswahl des passenden Event Driven Stacks wirklich ankommt – aus Entwicklersicht, nicht aus Marketingsicht
- Performance, Skalierbarkeit und Latenz: Was jeder Stack in der Praxis wirklich leistet (und wo die Mythen liegen)
- Welche Fehler in der Event Driven Architektur Projekte regelmäßig killen – und wie du sie verhinderst
- Schritt-für-Schritt-Checkliste: So wählst du deinen Event Driven Stack richtig aus
- Warum Cloud-native vs. Open Source nicht nur eine Kostenfrage ist
- Was 2025 beim Thema Event Driven Architecture wirklich zählt – und wie du dich gegen die Konkurrenz behauptest

Event Driven Architektur ist der feuchte Traum jedes CTOs, der nachts von Microservices und „echter“ Skalierbarkeit schwärmt. Doch in der Realität scheitern 90 Prozent der Projekte spätestens dann, wenn es um saubere Event-Definitionen, persistente Event Stores und den richtigen Stack geht. Wer glaubt, das nächste Kafka-Cluster sei die Antwort auf alle Probleme, hat das Thema nicht verstanden. Hier geht’s nicht um Buzzwords. Es geht um Systemdesign, Durchsatz, Latenz, Konsistenz und vor allem: um die Fähigkeit, Fehler zu vermeiden, bevor sie dich die nächste Million kosten. Zeit, den Event Driven Stack wirklich zu verstehen – und zwar bis ins letzte Bit.

# Was ist ein Event Driven Stack wirklich? – Architektur, Komponenten, Buzzword-Bingo

Ein Event Driven Stack ist kein Produkt, kein Framework und schon gar kein Plug-and-Play-Modul. Es ist ein Architekturkonzept, das Systeme darauf auslegt, auf Ereignisse (Events) in Echtzeit oder Near-Realtime zu reagieren. Die Idee: Statt Request/Response-Overkill setzt du auf lose gekoppelte Microservices, die über Events miteinander kommunizieren. Klingt cool? Ist es auch – wenn man’s richtig macht.

Die typischen Komponenten eines Event Driven Stacks:

- Event Producer: Services oder Anwendungen, die Events erzeugen und in einen Broker schicken
- Message Broker/Event Bus: Der zentrale Router, der Events empfängt, speichert und an Subscriber verteilt – hier kommen Kafka, RabbitMQ, NATS oder EventBridge ins Spiel
- Event Consumer: Services, die auf bestimmte Events reagieren und Aktionen auslösen
- Event Store: Persistente Speicherung aller Events, häufig als „Single

Source of Truth“ für Auditing und Replaying
- Schema Registry: Verwaltung von Event-Schemas, damit Producer und Consumer dieselbe Sprache sprechen

Soweit die Theorie. In der Praxis stolpern die meisten schon bei der sauberen Definition von Events (was ist ein Event, was ein Command?), der Konsistenzsicherung und dem Zusammenspiel aus Broker, Store und Processing Layer. Wer hier auf halbgare Lösungen setzt, produziert Chaos statt Skalierbarkeit.

Das größte Missverständnis: Ein Event Driven Stack bedeutet nicht automatisch „asynchron und skalierbar“. Ohne saubere Architektur und die richtige Tool-Auswahl wird aus dem Traum schnell ein Wartungs-Albtraum mit Race Conditions, Message Loss und Debugging-Hölle.

# Kafka, RabbitMQ, AWS EventBridge, NATS – Der große Vergleich der Event Driven Technologien

Jetzt wird’s ernst. Jeder kennt die Namen, aber kaum einer versteht die wirklichen Unterschiede. Die Wahl des Event Driven Stacks entscheidet über Durchsatz, Latenz, Wartbarkeit und die Zukunftsfähigkeit deiner Systeme. Zeit für den tabulosen Tech-Vergleich der Big Player:

- Apache Kafka
  - Stärken: Massiv skalierbar, hohe Durchsatzraten (Millionen Messages/sec), persistente Speicherung, Event Replaying per Default, starke Ökosystem-Tools (Kafka Streams, Connect, Schema Registry)
  - Schwächen: Komplexes Setup, anspruchsvolles Cluster-Management, Ressourcenfresser, hohe Latenz bei kleinen Nachrichten, nicht für klassische Queueing-Patterns gebaut
  - Use Case: Echtzeit-Analytics, Event Sourcing, Big Data Pipelines, Auditing
- RabbitMQ
  - Stärken: Einfaches Setup, flexible Routing-Patterns (Fanout, Topic, Direct), niedrige Latenz, gute Integration in klassische Message Queues
  - Schwächen: Kein persistentes Event Store-Konzept, limitiertes Event Replaying, Performance-Grenzen bei sehr hoher Last, weniger geeignet für Big Data
  - Use Case: Task-Queues, Microservice-Kommunikation, klassische Message-Patterns
- AWS EventBridge
  - Stärken: Voll gemanagt, nahtlose Integration in AWS Services,

beliebig skalierbar, Serverless, Pay-per-Use
    - Schwächen: Vendor Lock-in, komplexe Preisstruktur, begrenzte Latenz- und Durchsatzgarantien, Black-Box-Charakter, Debugging oft eine Qual
    - Use Case: Cloud-native Architekturen, Serverless Event Processing, Integration zwischen AWS Services
- NATS
    - Stärken: Ultraleicht, extrem geringe Latenz (Millisekunden-Bereich), einfache Installation, Cluster- und JetStream-Extension für Event Streaming
    - Schwächen: Weniger Features, kein vollwertiger persistenter Event Store (ohne JetStream), kleinere Community, weniger Enterprise-Support
    - Use Case: IoT, Edge Computing, High Performance Microservices mit Fokus auf Geschwindigkeit

Und dann gibt es noch Exoten wie Apache Pulsar, Google Pub/Sub oder Redpanda. Jeder Stack hat seine Daseinsberechtigung, aber: Kein Stack kann alles gleichzeitig. Wer Kafka für klassische Queue-Aufgaben nutzt, verschwendet Ressourcen. Wer EventBridge für kritische Audit-Trails einsetzt, läuft ins Risiko, weil Replaying und Persistenz nicht garantiert sind. Kurz: Stack-Auswahl ist kein Religionsthema, sondern eine Frage von Anforderungen, Use Case und Skillset.

# Event Driven Stack Auswahl: Worauf du wirklich achten musst (und was Marketingleute dir verschweigen)

Die perfekte Event Driven Architektur gibt es nicht, aber den perfekten Stack für deinen Use Case. Leider werden die wichtigsten Fragen oft ignoriert, weil sie unbequem sind – oder weil sie tiefes technisches Verständnis erfordern. Hier die Kriterien, die wirklich zählen:

- Durchsatz und Latenz: Muss dein System Millionen von Events pro Sekunde verarbeiten, oder reichen hundert pro Minute? Kafka rockt bei Durchsatz, NATS bei Latenz.
- Persistenz und Replaying: Brauchst du einen Event Store, um Events später erneut zu verarbeiten oder zu auditieren? Nur Kafka (und mit JetStream auch NATS) liefern das wirklich out-of-the-box.
- Fehlerhandling und Delivery Guarantees: Genau-once-Delivery ist das heilige Gral, aber die meisten Stacks bieten nur At-least-once oder Best-effort. Prüfe, was du wirklich brauchst – und was dein Stack kann.
- Skalierbarkeit und Wartbarkeit: Wie viele Nodes, wie viel Netzwerk-Traffic, wie aufwendig ist das Monitoring? Ein Kafka-Cluster ist kein Hobby-Projekt, RabbitMQ ist bei hoher Last schnell am Limit.

- Integrationen und Community: Gibt es fertige Connectors, SDKs, Monitoring-Tools? Wie schnell findest du Hilfe im Notfall?
- Cloud-native vs. Self-Hosted: Managed Services wie EventBridge nehmen dir Arbeit ab, kosten aber Flexibilität und Kontrolle. Open Source wie NATS gibt dir alles – inklusive Verantwortung.

Der Kardinalfehler: Viele Teams wählen „den, den alle nehmen“, statt ihren Stack auf den eigenen Anwendungsfall zuzuschneiden. Die Folge: Stack-Overkill oder Featurelücken, die dich Monate später einholen. Wer sich nicht brutal ehrlich mit Anforderungen und Know-how auseinandersetzt, baut sich einen Event Driven Frankenstein zusammen.

Das Mantra: Technologie ist kein Selbstzweck. Jeder Stack hat technische Limits – und die meisten werden in Marketing-PDFs verschwiegen. Wer sich von Vendor-Versprechen blenden lässt, zahlt mit Performance, Stabilität und am Ende mit dem eigenen Job.

# Performance, Skalierbarkeit, Latenz: Was die Event Driven Stacks wirklich leisten

Jetzt geht’s ans Eingemachte. Jeder Event Driven Stack wird mit sagenhaften Durchsatzraten, minimaler Latenz und unbegrenzter Skalierbarkeit beworben. Die Realität sieht anders aus – und hängt von deinem konkreten Setup, Netzwerk, Hardware und Use Case ab. Zeit, die Marketingblasen platzen zu lassen:

Kafka: Theoretisch mehrere Millionen Events pro Sekunde, aber nur bei optimalem Hardware-Setup und Tuning. Latenzen von 2–20ms, aber unter hoher Last und kleinen Messages kann’s schnell in den dreistelligen Bereich gehen. Skalierung horizontal, aber teuer im Betrieb und Monitoring. Ohne Expertenwissen: Katastrophe vorprogrammiert.

RabbitMQ: Schnell bei kleiner bis mittlerer Last, Latenzen oft unter 10ms. Doch bei zigtausenden Messages pro Sekunde geht RabbitMQ die Puste aus. Persistenz kostet Performance, komplexe Routing-Patterns erschweren das Debugging.

EventBridge: Skalierung in der AWS-Cloud quasi unbegrenzt – solange das Konto mitmacht. Latenzen meist im niedrigen Sekundenbereich, für Echtzeit kritisch. Debugging schwierig, Black-Box-Charakter. Für Auditing und Compliance ein Risiko.

NATS: Der Latenz-König – Millisekunden sind Standard, Durchsatz aber limitiert durch Storage und Netzwerk. JetStream bringt Persistenz, aber auf Kosten der Einfachheit. Perfekt für „Fire and Forget“, weniger für Big Data.

Wichtige Faustregel: Die allermeisten Probleme entstehen *nicht* durch den Stack, sondern durch falsche Architekturentscheidungen. Wer Kafka mit 100

Consumer Groups pro Topic betreibt, wundert sich über Performance-Kollaps. Wer RabbitMQ für Big Data missbraucht, bekommt Message Loss. Und wer EventBridge als Event Store missversteht, hat das nächste Compliance-Audit eigentlich schon verloren.

# Fehler in Event Driven Architekturen – und wie du sie gnadenlos vermeidest

Die Liste der Fehler ist lang, aber die Klassiker wiederholen sich in jedem zweiten Projekt. Wer sie kennt, kann sie verhindern – und spart sich Monate an Debugging, Downtime und Datenverlust.

- Keine saubere Event-Definition: Unklare Events führen zu doppelter Verarbeitung, Inkonsistenzen und Debugging-Hölle. Jedes Event braucht Versionierung, Schemas und klare Payloads.
- Fehlendes Error Handling: Events verschwinden im Nirvana, weil Dead Letter Queues fehlen oder falsch konfiguriert sind. At-least-once-Delivery ist Standard, aber idempotente Consumer sind Pflicht.
- Fehlende Monitoring- und Tracing-Tools: Ohne Observability ist jeder Event Driven Stack eine Black Box. Nutze Distributed Tracing (OpenTelemetry, Jaeger) und Metriken (Prometheus, Grafana).
- Event Store vs. Message Queue verwechselt: Wer Messaging und Event Sourcing nicht trennt, produziert Chaos. Kafka ist kein Task-Queue-Ersatz, RabbitMQ kein Big Data Store.
- Vendor Lock-in unterschätzt: Cloud-native Lösungen wie EventBridge sind bequem, aber die Migration kann zum Desaster werden.

Und das Wichtigste: Teste mit echten Lastszenarien. Viele Stacks brechen erst unter Realbedingungen ein – und dann ist es zu spät. Wer nicht regelmäßig Chaos Engineering und Lasttests fährt, betreibt Event Driven Roulette.

# Schritt-für-Schritt-Checkliste: So findest du den richtigen Event Driven Stack

- 1. Anforderungen definieren: Wie viele Events pro Sekunde? Wie wichtig ist Persistenz? Wie kritisch sind Latenzen?
- 2. Event- und Messaging-Patterns klären: Brauchst du Event Sourcing, klassische Queues, Pub/Sub, oder alles zusammen?
- 3. Skillset und Team-Know-how prüfen: Bringt dein Team Kafka-Cluster-Management mit? Oder ist Managed Service die bessere Wahl?
- 4. Integrationen und Ökosystem-Check: Gibt es fertige Connectors für

Datenbanken, Monitoring, Alerting?
- 5. Testen, Testen, Testen: Lasttests, Failure Scenarios, Recovery – alles vor dem Go Live durchspielen.
- 6. Monitoring und Observability einbauen: Ohne Telemetrie keine Kontrolle – von Anfang an einplanen.
- 7. Kosten und Betrieb realistisch kalkulieren: Was kostet Cluster-Betrieb, Cloud-Events, Support?
- 8. Dokumentation und Event-Schemas pflegen: Ohne Schema Registry und Versionierung ist jeder Stack nach sechs Monaten Legacy.

# Fazit: Event Driven Architektur 2025 – Mehr als nur Stack-Entscheidung

Wer 2025 ein Event Driven System bauen will, muss mehr draufhaben als ein paar Buzzwords und ein Kafka-Docker-Compose. Der Event Driven Stack ist das technische Rückgrat moderner Microservices – aber nur, wenn er zum Use Case, zum Team und zur Infrastruktur passt. Es gibt keinen perfekten Stack, nur die perfekte Lösung für deine Anforderungen. Wer blind dem Hype folgt, zahlt mit Downtime, Datenverlust und Frustration.

Die Moral von der Geschichte: Nicht der Stack entscheidet über Erfolg oder Scheitern, sondern Architektur, Know-how und die Bereitschaft, brutal ehrlich auf technische Limits zu schauen. Wer Event Driven wirklich meistern will, muss sich tief in Technologie, Patterns und Monitoring reinknien – und darf sich nie von Marketing-Sprech blenden lassen. Willkommen im echten Leben der Event Driven Architekturen. Willkommen bei 404.