Task: PSR-94 - R&D to select Event Bus

Compare Message queues: Apache Kafka, NATS, RabbitMQ

- Reliability
- Feature Rich
- In-premises setup
- Throughput: Performance
- High Availability
-

Factor	**Apache Kafka**	**NATS**	**RabbitMQ**
Persisting event/messages Feature	Kafka persists all messages by design up to a configured timeout per topic	NATS is a fire-and-forget messaging system. NATS doesn't do persistent messaging; if you're offline, you don't get the message	RabbitMQ evicts messages from storage as soon as consumers successfully consume them. This behavior cannot be modified
Client for C#, Python, and Java	yes	yes	Yes (MassTransitframework support for .net framework for python)
standardized protocols such as AMQP, MQTT, STOMP	no	no	yes
Monitoring:web-based dashboard	yes	NATS Surveyor, Prometheus NATS Exporter. no web-based dashboard	yes
High Availability			RabbitMQ's high availability support is terrible. (You can do it by mirroring.)

Light-weight	no	yes	no
Maturity and community	Mature & broad community		
Message ordering	Kafka provides a reliable ordering guarantee on message processing. Kafka guarantees that all messages sent to the same topic partition are processed in-order.		RabbitMQ - If a single message consumer, it receives messages in order. If not no guarantee.
Message routing	Kafka does not allow consumers to filter messages in a topic before polling them. A subscribed consumer receives all messages in a partition without exception.		RabbitMQ can route messages to subscribers of a message exchange based on subscriber-defined routing rules
Message timing	It writes messages to partitions as they arrive, where they are immediately available for consumers to consume. Kafka provides no TTL mechanism for messages		RabbitMQ supports delayed/scheduled messages
Message retention	Kafka persists all messages by design up to a configured timeout per topic.		RabbitMQ evicts messages from storage as soon as consumers successfully consume them. This behavior cannot be modified. It is part of almost all message brokers' design.

Fault handling	Kafka does not provide any such mechanisms out of the box. With Kafka, it is up to us to provide and implement message retry mechanisms at the application level.		RabbitMQ provides tools such as delivery retries and dead-letter exchanges (DLX) to handle message processing failures.
Scale	Kafka is generally considered to have better performance than RabbitMQ. Kafka uses sequential disk I/O to boost performance. Large Kafka deployments can commonly handle hundreds of thousands of messages per second, and even millions of messages per second.		Typical RabbitMQ deployments include three to seven node clusters that do not necessarily optimally divide the load between queues. These typical clusters can usually expect to handle a load of several tens of thousands of messages per second.
Consumer complexity	Kafka uses a dumb-broker and smart-consumer approach		RabbitMQ uses a smart-broker and dumb-consumer approach

RabbitMQ is preferable when we need:

1. Advanced and flexible routing rules.
2. Message timing control (controlling either message expiry or message delay).
3. Advanced fault handling capabilities, in cases when consumers are more likely to fail to process messages (either temporarily or permanently).
4. Simpler consumer implementations.

Kafka is preferable when we require:

1. Strict message ordering.
2. Message retention for extended periods, including the possibility of replaying past messages.
3. The ability to reach a high scale when traditional solutions do not suffice.