

Source and Mirror Streams



- [General behavior](#)
 - [Source specific](#)
 - [Mirror specific](#)
 - [Expected behavior in edge conditions](#)
 - [WorkQueue/Interest retention](#)
 - ["Bring your own" sourcing consumer](#)
-

When a stream is configured with a `source` or `mirror`, it will automatically and asynchronously replicate messages from the origin stream.

`source` or `mirror` are designed to be robust and will recover from a loss of connection. They are suitable for geographic distribution over high latency and unreliable connections. E.g. even a leaf node starting and connecting intermittently every few days will still receive or send messages over the source/mirror link.

Another use case is when [connecting streams cross-account](#).

There are several options available when declaring the configuration.

- `Name` - Name of the origin stream to source messages from.
- `StartSeq` - An optional start sequence of the origin stream to start mirroring from.
- `StartTime` - An optional message start time to start mirroring from. Any messages that are equal to or greater than the start time will be included.
- `FilterSubject` - An optional filter subject which will include only messages that match the subject, typically including a wildcard. Note, this cannot be used with `SubjectTransforms`.
- `SubjectTransforms` - An optional set of [subject transforms](#) to apply when sourcing messages from the origin stream. Note, in this context, the `Source` will act as a filter on the origin stream and the `Destination` can optionally be provided to apply a transform. Since multiple subject transforms can be used, disjoint subjects can be sourced from the origin stream while maintaining the order of the messages. Note, this cannot be used with `FilterSubject`.
- `Domain` - An optional JetStream domain of where the origin stream exists. This is commonly used in a hub cluster and leafnode topology.

The stream using a source or mirror configuration can have its own retention policy, replication, and storage type.





- Changes to the stream using source or mirror, e.g. deleting messages or publishing, do not reflect back on the origin stream from which the data was received.
- Deletes in the origin stream are NOT replicated through a `source` or `mirror` agreement.



`Sources` is a generalization of the `Mirror` and allows for sourcing data from one or more streams concurrently.

If you require the target stream to act as a read-only replica:

- Configure the stream without listen subjects **or**
- Temporarily disable the listen subjects through client authorizations.

General behavior

- All configurations are made on the receiving side. The stream from which data is sourced and mirrored does not need to be configured. No cleanup is required on the origin side if the receiver disappears.
- A stream can be the origin (source) for multiple streams. This is useful for geographic distribution or for designing "fan out" topologies where data needs to be distributed reliable to a large number (up to millions) of client connections.
- Leaf nodes and leaf node domains are explicitly supported through the `API prefix`

Source specific

A stream defining `Sources` is a generalized replication mechanism and allows for sourcing data from **one or more streams** concurrently. A stream with sources can still act as a regular stream allowing direct write/publish by local clients to the stream. Essentially the source streams and local client writes are aggregated into a single interleaved stream.

Combined with subject transformation and filtering sourcing allows to design sophisticated data distribution architectures.



Sourcing messages does not retain sequence numbers. But it retain the in stream sequence of messages . Between streams sourced to the same target, the sequence of messages is undefined.

Mirror specific

A mirror can source its messages from **exactly one stream** and a clients can not directly write to the mirror. Although messages cannot be published to a mirror directly by clients, messages can be deleted on-demand (beyond the retention policy), and consumers have all capabilities available on regular streams.

“ ”

- Mirrored messages retains the sequence numbers and timestamps of the origin stream.
- Mirrors can be used for (geographic) load distribution with the `MirrorDirect` stream attribute. See: <https://docs.nats.io/nats-concepts/jetstream/streams#configuration>

Expected behavior in edge conditions

- Source and mirror contracts are designed with one-way (geographic) data replication in mind. Neither configuration provides a full synchronization between streams, which would include deletes or replication of other stream attributes.
- The content of the stream from which a source or mirror is drawn needs to be reasonable stable. Quickly deleting messages after publishing them may result in inconsistent replication due to the asynchronous nature of the replication process.
- Sources and Mirror try to be efficient in replicating messages and are lenient towards the source/mirror origin being unreachable (event for extended periods of time), e.g. when using leaf nodes, which are connected intermittently. For sake of efficiency the recovery interval in case of a disconnect is 10-20s.
- Mirror and source agreements do not create a visible consumer in the origin stream.

WorkQueue/Interest retention

“ ”

Stream sourcing and mirroring for WorkQueue or Interest streams are supported since version 2.14.

When using stream sourcing/mirroring on a WorkQueue or Interest stream, the consumer used is automatically 'upgraded' to a durable consumer. This consumer is visible to the user and can be monitored or paused to temporarily stop the sourcing from the sourced-side.

“ ”

On versions prior to 2.14, sourcing from a WorkQueue/Interest stream is not supported. JetStream does not forbid this configuration, but the behavior is undefined.

"Bring your own" sourcing consumer

Starting in version 2.14, you can "bring your own" sourcing consumer. This pre-created consumer can then be used for stream sourcing/mirroring, instead of the server automatically creating it based on the stream configuration. Useful for more flexible consumer configuration, as well as additional monitoring or operational capabilities, or generally when you're not in control of "the other side" and are required to do this from a security point of view.

The following options are used to "bring your own" sourcing consumer:

- `Name` - Name of the origin stream to source messages from.
- `Consumer.Name` - The consumer name used for sourcing.
- `Consumer.DeliverSubject` - The consumer deliver subject that messages are delivered to.

Other options like `StartSeq`, `StartTime`, `FilterSubject`, etc. are not supported when using this approach. Instead, these settings should be configured directly on the consumer.

This consumer is required to have `AckFlowControl` as its ack policy, meaning the sourced-side will only receive acknowledgements until after the sourcing-side has persisted the messages.

#3

cocobook-sync 2026-10-28 11:10:22 UTC

cocobook-sync 2026-10-29 01:42:33 UTC