

MQTT



- [When to Use MQTT](#)
 - [JetStream Requirements](#)
 - [MQTT Topics and NATS Subjects](#)
 - [MQTT Wildcards](#)
 - [NATS Wildcards in MQTT Topics](#)
 - [MQTT topics and NATS permissions](#)
 - [Communication Between MQTT and NATS](#)
 - [QoS 1 and 2 Redeliveries](#)
 - [Max Ack Pending](#)
 - [Sessions](#)
 - [Retained Messages](#)
 - [Clustering](#)
 - [Connections with Same Client ID](#)
 - [Retained Messages](#)
 - [Limitations](#)
 - [See Also](#)
-

Supported since NATS Server version 2.2

NATS follows as closely as possible to the MQTT v3.1.1 [specification](#). Refer to the [MQTT implementation overview](#) in the nats-server repo.

NATS supports MQTT QoS: 0, 1 and 2

When to Use MQTT

MQTT support in NATS is intended to be an enabling technology allowing users to leverage existing investments in their IoT deployments. Updating software on the edge or endpoints can be onerous and risky, especially when embedded applications are involved.

In greenfield IoT deployments, when possible, we prefer NATS extended out to endpoints and devices for a few reasons. There are significant advantages with security and observability when using a single technology end to end. Compared to MQTT, NATS is nearly as lightweight in terms of protocol bandwidth and maintainers supported clients efficiently utilize resources so we consider NATS to be a good choice to use end to end, including use on resource constrained devices.

In existing MQTT deployments or in situations when endpoints can only support MQTT, using a NATS server as a drop-in MQTT server replacement to securely connect to a remote NATS cluster or supercluster is compelling. You can keep your existing IoT investment and use NATS for secure, resilient, and scalable access to your streams and services.

JetStream Requirements

For an MQTT client to connect to the NATS server, the user's account must be JetStream enabled. This is because persistence is needed for the sessions and retained messages since even retained messages of QoS 0 are persisted.

MQTT Topics and NATS Subjects

MQTT Topics are similar to NATS Subjects, but have distinctive differences.

MQTT topic uses "/" as a level separator. For instance `foo/bar` would translate to NATS subject `foo.bar`. But in MQTT, `/foo/bar/` is a valid subject, which, if simply translated, would become `.foo.bar.`, which is NOT a valid NATS Subject.

NATS Server will convert an MQTT topic following those rules:

MQTT character	NATS character(s)	Topic (MQTT)	Subject (NATS)
/ between two levels	.	foo/bar	foo.bar
/ as first level	/.	/foo/bar	/.foo.bar
/ as last level	./	foo/bar/	foo.bar./
/ next to another	./	foo//bar	foo./.bar
/ next to another	/.	//foo/bar	./.foo.bar
.	// (see note below)	foo.bar	foo//bar
	Not Supported	foo bar	Not Supported

Note: Prior to NATS Server v2.10.0, the character . was not supported. At version v2.10.0 and above, the character . will be translated to //.

As indicated above, if an MQTT topic contains the character `#` (or `.` prior to v2.10.0), NATS will reject it, causing the connection to be closed for published messages, and returning a failure code in the SUBACK packet for a subscriptions.

MQTT Wildcards

As in NATS, MQTT wildcards represent either multi or single levels. As in NATS, they are allowed only for subscriptions, not for published messages.

MQTT Wildcard	NATS Wildcard
#	>
+	*

The wildcard `#` matches any number of levels within a topic, which means that a subscription on `foo/#` would receive messages on `foo/bar`, or `foo/bar/baz`, but also on `foo`. This is not the case in NATS where a subscription on `foo.>` can receive messages on `foo/bar` or `foo/bar/baz`, but not on `foo`. To solve this, NATS Server will create two subscriptions, one on `foo.>` and one on `foo`. If the MQTT subscription is simply on `#`, then a single NATS subscription on `>` is enough.

The wildcard `+` matches a single level, which means `foo/+` can receive message on `foo/bar` or `foo/baz`, but not on `foo/bar/baz` nor `foo`. This is the same with NATS subscriptions using the wildcard `*`. Therefore `foo/+` would translate to `foo.*`.

NATS Wilcards in MQTT Topics

MQTT treats `*` and `>` as ordinary literal characters. The NATS Server does not escape these characters during MQTT topic to NATS subject conversion. If either character becomes a complete NATS subject token after `/` is converted to `.`, NATS interprets it using normal NATS wildcard rules.

Topic (MQTT)	Subject (NATS)	Behavior
foo*/bar	foo*.bar	literal
foo>/bar	foo>.bar	literal
foo/*/bar	foo.*.bar	wildcard
foo/>	foo.>	wildcard

MQTT topics and NATS permissions

MQTT topics are converted to NATS subjects before authorization checks are applied. Permission entries for MQTT users should therefore be written as NATS literal or wildcard subjects.

Communication Between MQTT and NATS

When an MQTT client creates a subscription on a topic, the NATS server creates the similar NATS subscription (with conversion from MQTT topic to NATS subject) so that the interest is registered in the cluster and known to any NATS publishers.

That is, say an MQTT client connects to server "A" and creates a subscription of `foo/bar`, server "A" creates a subscription on `foo.bar`, which interest is propagated as any other NATS subscription. A publisher connecting anywhere in the cluster and publishing on `foo.bar` would cause server "A" to deliver a QoS 0 message to the MQTT subscription.

This works the same way for MQTT publishers. When the server receives an MQTT publish message, it is converted to the NATS subject and published, which means that any matching NATS subscription will receive the MQTT message.

If the MQTT subscription is QoS1 and an MQTT publisher publishes an MQTT QoS1 message on the same or any other server in the cluster, the message will be persisted in the cluster and routed and delivered as QoS 1 to the MQTT subscription.

QoS 1 and 2 Redeliveries

When the server delivers a QoS 1 or 2 message to a QoS 1 or 2 subscription, it will keep the message until it receives the PUBACK for the corresponding packet identifier. If it does not receive it within the "ack_wait" interval, that message will be resent.

Max Ack Pending

This is the amount of QoS 1 or 2 messages the server can send to a subscription without receiving any PUBACK for those messages. The maximum value is 65535.

The total of subscriptions' `max_ack_pending` on a given session cannot exceed 65535. Attempting to create a subscription that would bring the total above the limit would result in the server returning a failure code in the SUBACK for this subscription.

Due to how the NATS server handles the MQTT "`#`" wildcard, each subscription ending with "`#`" will use 2 times the `max_ack_pending` value.

Sessions

NATS Server will persist all sessions, even if they are created with the "clean session" flag, meaning that sessions only last for the duration of the network connection between the client and the server.

A session is identified by a client identifier. If two connections try to use the same client identifier, the server, per specification, will close the existing connection and accept the new one.

If the user incorrectly starts two applications that use the same client identifier, this would result in a very quick flapping if the MQTT client has a reconnect feature and quickly reconnects.

To prevent this, the NATS server will accept the new session and will delay the closing of the old connection to reduce the flapping rate.

Detection of the concurrent use of sessions also works in cluster mode.

Retained Messages

When a server receives an MQTT publish packet with the RETAIN flag set (regardless of its QoS), it stores the application message and its QoS, so that it can be delivered to future subscribers whose subscriptions match its topic name.

When a new subscription is established, the last retained message, if any, on each matching topic name will be sent to the subscriber.

A PUBLISH Packet with a RETAIN flag set to 1 and a payload containing zero bytes will be processed as normal and sent to clients with a subscription matching the topic name. Additionally any existing retained message with the same topic name will be removed and any future subscribers for the topic will not receive a retained message.

Clustering

NATS supports MQTT in a NATS cluster. The replication factor is automatically set based on the size of the cluster.

Connections with Same Client ID

If a client is connected to a server "A" in the cluster and another client connects to a server "B" and uses the same client identifier, server "A" will close its client connection upon discovering the use of an active client identifier.

Users should avoid this situation as this is not as easy and immediate as if the two applications are connected to the same server.

There may be cases where the server will reject the new connection if there is no safe way to close the existing connection, such as when it is in the middle of processing some MQTT packets.

Retained Messages

Retained messages are stored in the cluster and available to any server in the cluster. However, this is not immediate and if a producer connects to a server and produces a retained message and another connection connects to another server and starts a matching subscription, it may not receive the retained message if the server it connects to has not yet been made aware of this retained message.

In other words, retained messages in clustering mode is best-effort, and applications that rely on the presence of a retained message should connect on the server that produced them.

Limitations

- NATS messages published to MQTT subscriptions are always delivered as QoS 0 messages.
- MQTT published messages on topic names containing "````" or ".\" characters will cause the connection to be closed. Presence of those characters in MQTT subscriptions will result in error code in the SUBACK packet.
- MQTT wildcard # may cause the NATS server to create two subscriptions.
- MQTT concurrent sessions may result in the new connection to be evicted instead of the existing one.
- MQTT retained messages in clustering mode is best effort.

See Also

[Replace your MQTT Broker with NATS Server](#)

📄 #1

📄 cocobook-sync 📄 2026-10-28 11:14:21 UTC

📄 cocobook-sync 📄 2026-10-28 11:14:21 UTC