

Leaf Nodes



- [LeafNode Configuration Tutorial](#)
 - [Leaf Node Example Using a Remote Global Service](#)
 - [Leaf Authorization](#)
 - [TLS-first Handshake](#)
-

A *Leaf Node* extends an existing NATS system of any size, optionally bridging both operator and security domains. A leafnode server will transparently route messages as needed from local clients to one or more remote NATS system(s) and vice versa. The leaf node authenticates and authorizes clients using a local policy. Messages are allowed to flow to the cluster or into the leaf node based on leaf node connection permissions of either.

Leaf nodes are useful in IoT and edge scenarios and when the local server traffic should be low RTT and local unless routed to the super cluster. NATS' queue semantics are honored across leaf connections by serving local queue consumers first.

- Clients to leaf nodes authenticate locally (or just connect if authentication is not required)
- Traffic between the leaf node and the cluster assumes the restrictions of the user configuration used to create the leaf connection.
 - Subjects that the user is allowed to publish are exported to the cluster.
 - Subjects the user is allowed to subscribe to, are imported into the leaf node.

Unlike [cluster](#) or [gateway](#) nodes, leaf nodes do not need to be reachable themselves and can be used to explicitly configure any acyclic graph topologies.

If a leaf node connects to a cluster, it is recommended to configure it with knowledge of **all** *seed servers* and have **each** *seed server* accept connections from leaf nodes. Should the remote cluster's configuration change, the discovery protocol will gossip peers capable of accepting leaf connections. A leaf node can have multiple remotes, each connecting to a different cluster. Each URL in a remote needs to point to the same cluster. If one node in a cluster is configured as leaf node, **all** nodes need to. Likewise, if one server in a cluster accepts leaf node connections, **all** servers need to.

“ Leaf Nodes are an important component as a way to bridge traffic between local NATS servers you control and servers that are managed by a third-party.

[Synadia's NGS](#) allows accounts to use leaf nodes, but gain accessibility to the global network to inexpensively connect geographically distributed servers or small clusters.

[LeafNode Configuration Options](#)

LeafNode Configuration Tutorial

The main server is just a standard NATS server. Clients to the main cluster are just using token authentication, but any kind of authentication can be used. The server allows leaf node connections at port 7422 (default port):

```
leafnodes {
  port: 7422
}
accounts: {
  app: {
    users: [
      {user: appuser, password: s3cr3t},
      {user: leafuser, password: s3cr3t}
    ]
  }
}
```

Start the server:

```
nats-server -c /tmp/server.conf
```

Output extract

```
...
[5774] 2019/12/09 11:11:23.064276 [INF] Listening for leafnode connections on 0.0.0.0:7422
...
```

We create a replier on the server to listen for requests on 'q', which it will aptly respond with '42':

```
nats reply -s nats://appuser:s3cr3t@localhost q 42
```

The leaf node, allows local clients to connect to through port 4111, and doesn't require any kind of authentication. The configuration specifies where the remote cluster is located, and specifies how to connect to it (just a simple token in this case):

```
listen: "127.0.0.1:4111"
leafnodes {
  remotes = [
    {
      url: "nats://leafuser:s3cr3t@localhost"
    },
  ]
}
```

In the case where the remote leaf connection is connecting with `tls`:

```
listen: "127.0.0.1:4111"
leafnodes {
  remotes = [
    {
      url: "tls://leafuser:s3cr3t@localhost"
    },
  ]
}
```

Note the leaf node configuration lists a number of `remotes`. The `url` specifies the port on the server where leaf node connections are allowed.

Start the leaf node server:

```
nats-server -c /tmp/leaf.conf
```

Output extract

```
....
[3704] 2019/12/09 09:55:31.548308 [INF] Listening for client connections on 127.0.0.1:4111
...
[3704] 2019/12/09 09:55:31.549404 [INF] Connected leafnode to "localhost"
```

Connect a client to the leaf server and make a request to 'q':

```
nats req -s nats://127.0.0.1:4111 q ""
```

```
Published [q] : ''
```

```
Received [_INBOX.Ua820JamRdWof5FBoiKaRm.gZhJP6RU] : '42'
```

Leaf Node Example Using a Remote Global Service

In this example, we connect a leaf node to Synadia's [NGS](#). Leaf nodes are supported on free developer and paid accounts. To use NGS, ensure that you've signed up and have an account loaded on your local system. It takes less than 30 seconds to grab yourself a free account to follow along if you don't have one already!

The `nsc` tool can operate with many accounts and operators, so it's essential to make sure you're working with the right operator and account. You can set the account using the `nsc` tool like below. The `DELETE_ME` account is used as an example, which is registered with NGS as a free account.

```
> nsc env -a DELETE_ME
> nsc describe account
```

Account Details	
Name	DELETE_ME
Account ID	ABF3NX7FJLDCU05QXBH56PV6EU4PR5HFCUJBXAG57AKSDUBTG0RDFOLI
Issuer ID	ODSKBNDIT3LTZWFSRAW0BXS7VZCDQVU6TBJX3TQGYXUWURU46ANJJS4
Issued	2023-03-02 18:18:42 UTC
Expires	
Max Connections	10
Max Leaf Node Connections	Not Allowed
Max Data	1.0 GB (1000000000 bytes)
Max Exports	2
Max Imports	7
Max Msg Payload	1.0 kB (1000 bytes)
Max Subscriptions	10
Exports Allows Wildcards	True
Disallow Bearer Token	False
Response Permissions	Not Set

Jetstream	Disabled	
+-----+-----+		
Exports	None	
+-----+-----+		

The `nsc` tool is aware of the account, so let's proceed to create a user for our example.

```
nsc add user leaftestuser
```

```
[ OK ] generated and stored user key
"UB5QBEU4LU70R26JEYSG27HH265QVUFGXYVBRD7SVKQJMEFSZTGFU62F"
[ OK ] generated user creds file "~/.nkeys/creds/synadia/leaftest/leaftestuser.creds"
[ OK ] added user "leaftestuser" to account "leaftest"
```

Let's craft a leaf node connection much like we did earlier:

```
leafnodes {
  remotes = [
    {
      url: "tls://connect.ngs.global"
      credentials: "/Users/alberto/.nkeys/creds/synadia/leaftest/leaftestuser.creds"
    },
  ]
}
```

The default port for leaf nodes is 7422, so we don't have to specify it.

Let's start the leaf server:

```
nats-server -c /tmp/ngs_leaf.conf
```

```
...
[4985] 2023/03/03 10:55:51.577569 [INF] Listening for client connections on 0.0.0.0:4222
...
[4985] 2023/03/03 10:55:51.918781 [INF] Connected leafnode to "connect.ngs.global"
```

Again, let's connect a replier, but this time to Synadia's NGS. NSC connects specifying the credentials file:

```
nsc reply q 42
```

And now let's make the request from the local host:

```
nats-req q ""
```

```
Published [q] : ''
```

```
Received [_INBOX.hgG0zVcVcyr4G5KBw0uyJw.uUYkEyKr] : '42'
```

Leaf Authorization

In some cases you may want to restrict what messages can be exported from the leaf node or imported from the leaf connection. You can specify restrictions by limiting what the leaf connection client can publish and subscribe to. See [NATS Authorization](#) for how you can do this.

TLS-first Handshake

As of NATS v2.10.0

Leafnode connections follow the model where when a TCP connection is created to the server, the server will immediately send an [INFO protocol message](#) in clear text. This INFO protocol provides metadata, including whether the server requires a secure connection.

Some environments prefer not to want a server that is configured to accept TLS connections for leafnodes having any traffic sent in clear text. It was possible to by-pass this using a websocket connection. However, if websocket is not desired, the accepting and remote servers can be [configured](#) to perform a TLS handshake before sending the INFO protocol message.

📄 #3

📄 cocobook-sync 📄 2026-10-28 11:13:51 UTC

📄 cocobook-sync 📄 2026-10-29 01:42:38 UTC