

JetStream on Leaf Nodes



- [Configuration](#)
 - [accounts.conf Imported by Both Servers](#)
 - [hub.conf](#)
 - [leaf.conf](#)
 - [Usage](#)
 - [Copying across domains via source or mirror](#)
 - [Cross account & domain import](#)
 - [Copying via source and mirror](#)
 - [Direct access of a durable pull consumer](#)
 - [See Also](#)
-

If you want to see a demonstration of the full range of this functionality, check out our [video](#)

One of the use cases for a NATS server configured as a [leaf node](#) is to provide a local NATS network even when the connection to a hub or the cloud is down. To support such a disconnected use case with JetStream, independent JetStream islands are also supported and available through the same NATS network.

The general issue with multiple, independent JetStreams, accessible from the same client is that you need to be able to tell them apart. As an example, consider a leaf node with a non-clustered JetStream in each server. You connect to one of them, but which JetStream responds when you use the JetStream API `$JS.API.>` ?

To disambiguate between servers, the option `domain` was added to the JetStream configuration block. When using it, follow these rules: Every server in a cluster and super cluster needs to have the same domain name. This means that domain names can only change between two servers if they are connected via a leaf node connection. As a result of this the JetStream API `$JS.API.>` will also be available under a disambiguated name `$JS.<domain>.API.>`. Needless to say, domain names need to be unique.

There are reasons to connect system accounts on either end of your leaf node connection. You probably don't want to connect your cloud and edge device system accounts, but you might connect them when the only reason keeping you from using a super cluster are firewall rules.

The benefits are:

- Monitoring of all connected nats-servers
- nats-account-resolver working on the entire network
- extended JetStream cluster

When `domain` is set, JetStream-related traffic on the system account is suppressed. This is what causes JetStream not to be extended.

In addition, traffic on `$JS.API.>` is also suppressed. This causes clients to use the local JetStream that is available in the nats-servers they are connected to. To communicate with another JetStream, that JetStream's unique domain specific prefix `$JS.<domain>.API` needs to be specified.

Please be aware that each domain is an independent name space. Meaning, inside the same account it is legal to use the same stream name in different domains.

Furthermore, regular message flow is not restricted. Thus, if the same subject is subscribed to by different streams in the same account in different domains, as long as the underlying leaf node was connected at the time, each stream will store the message. This can be resolved by using the same account but use different subjects in each domain or use different accounts in each domain or [isolate accounts](#) used in leaf nodes.

“ *Known issue:* if you have more than one JetStream enabled leaf node in a different cluster, the cluster you connect to also needs JetStream enabled and a domain set.

Known issue: when you intend to extend a central JetStream, by not supplying a domain name in leaf nodes, that central JetStream needs to be in clustered mode.

Configuration

Below is the config needed to connect two JetStream enabled servers via a leaf node connection. In the example, the system accounts are connected for demonstration purposes (you do not have to do that).

`accounts.conf` Imported by Both Servers

```
accounts {
  SYS: {
    users: [{user: admin, password: admin}]
  },
  ACC: {
    users: [{user: acc, password: acc}],
    jetstream: enabled
  }
}

system_account: SYS
```

hub.conf

To be started with `nats-server -c hub.conf`:

```
port: 4222
server_name: hub-server
jetstream {
  store_dir="./store_server"
  domain=hub
}
leafnodes {
  port: 7422
}
include ./accounts.conf
```

leaf.conf

To be started with `nats-server -c leaf.conf`:

```
port: 4111
server_name: leaf-server
jetstream {
  store_dir="./store_leaf"
  domain=leaf
}
leafnodes {
  remotes = [
```

```

{
  urls: ["nats://admin:admin@0.0.0.0:7422"]
  account: "SYS"
},
{
  urls: ["nats://acc:acc@0.0.0.0:7422"]
  account: "ACC"
}
]
}
include ./accounts.conf

```

Usage

Because the system account is connected, you can obtain the JetStream server report from both servers.

```
nats --server nats://admin:admin@localhost:4222 server report jetstream
```

JetStream									
Summary									
Server	Cluster	Domain	Streams	Consumers	Messages	Bytes	Memory	File	
API Req	API Err								
leaf-server	leaf-server	leaf	0	0	0	0 B	0 B	0 B	
0	0								
hub-server		hub	0	0	0	0 B	0 B	0 B	
0	0								
			0	0	0	0 B	0 B	0 B	
0	0								

Create a stream named `test` subscribing to subject `test` in the JetStream domain, the program is connected to. As a result, this stream will be created in the domain hub which is the domain of the server listening on `localhost:4222`.

```
nats --server nats://acc:acc@localhost:4222 stream add
```

```
? Stream Name test
? Subjects to consume test
? Storage backend file
? Retention Policy Limits
? Discard Policy Old
? Stream Messages Limit -1
? Per Subject Messages Limit -1
? Message size limit -1
? Maximum message age limit -1
? Maximum individual message size -1
? Duplicate tracking time window 2m
? Replicas 1
Stream test was created

Information for Stream test created 2021-06-28T12:52:29-04:00

Configuration:

    Subjects: test
Acknowledgements: true
    Retention: File - Limits
    Replicas: 1
    Discard Policy: Old
Duplicate Window: 2m0s
Maximum Messages: unlimited
    Maximum Bytes: unlimited
    Maximum Age: 0.00s
Maximum Message Size: unlimited
    Maximum Consumers: unlimited

State:
```

```
Messages: 0
Bytes: 0 B
FirstSeq: 0
LastSeq: 0
Active Consumers: 0
```

To create a stream in a different domain while connected somewhere else, just provide the `js-domain` argument. While connected to the same server as before, now the stream is created in `leaf`.

```
nats --server nats://acc:acc@localhost:4222 stream add --js-domain leaf
```

```
? Stream Name test
? Subjects to consume test
? Storage backend file
? Retention Policy Limits
? Discard Policy Old
? Stream Messages Limit -1
? Per Subject Messages Limit -1
? Message size limit -1
? Maximum message age limit -1
? Maximum individual message size -1
? Duplicate tracking time window 2m
? Replicas 1
Stream test was created
```

Information for Stream test created 2021-06-28T12:59:18-04:00

Configuration:

```
Subjects: test
Acknowledgements: true
Retention: File - Limits
Replicas: 1
Discard Policy: Old
Duplicate Window: 2m0s
Maximum Messages: unlimited
Maximum Bytes: unlimited
Maximum Age: 0.00s
```

Maximum Consumers: unlimited

Active Consumers: 0

```
nats --server nats://acc:acc@localhost:4222 pub test "hello world"
```

```
nats --server nats://acc:acc@localhost:4222 stream report --js-domain leaf
```

Stream Report							
Stream	Storage	Consumers	Messages	Bytes	Lost	Deleted	Replicas
test	File	0	1	45 B	0	0	

Stream Report

Stream	Storage	Consumers	Messages	Bytes	Lost	Deleted	Replicas
test	File	0	1	45 B	0	0	

Copying across domains via `source` or `mirror`

In order to copy a stream from one domain into another, specify the JetStream domain when creating a `mirror`. If you want to connect a leaf to the hub and get commands, even when the leaf node connection is offline, mirroring a stream located in the hub is the way to go.

```
nats --server nats://acc:acc@localhost:4222 stream add --js-domain hub --mirror test
```

```
? Stream Name backup-test-leaf
? Storage backend file
? Retention Policy Limits
? Discard Policy Old
? Stream Messages Limit -1
? Message size limit -1
? Maximum message age limit -1
? Maximum individual message size -1
? Replicas 1
? Adjust mirror start No
? Import mirror from a different JetStream domain Yes
? Foreign JetStream domain name leaf
? Delivery prefix
```

Stream backup-test-leaf was created

Information for Stream backup-test-leaf created 2021-06-28T14:00:43-04:00

Configuration:

```
Acknowledgements: true
  Retention: File - Limits
    Replicas: 1
  Discard Policy: Old
Duplicate Window: 2m0s
```

Maximum Messages: unlimited

Maximum Bytes: unlimited

Maximum Age: 0.00s

Maximum Message Size: unlimited

Maximum Consumers: unlimited

Mirror: test, API Prefix: \$JS.leaf.API, Delivery Prefix:

State:

Messages: 0

Bytes: 0 B

FirstSeq: 0

LastSeq: 0

Active Consumers: 0

Similarly, if you want to aggregate streams located in any number of leaf nodes use `source`. If the streams located in each leaf are used for the same reasons, it is recommended to aggregate them in the hub for processing via `source`.

```
nats --server nats://acc:acc@localhost:4222 stream add --js-domain hub --source test
```

```
? Stream Name aggregate-test-leaf
? Storage backend file
? Retention Policy Limits
? Discard Policy Old
? Stream Messages Limit -1
? Message size limit -1
? Maximum message age limit -1
? Maximum individual message size -1
? Duplicate tracking time window 2m
? Replicas 1
? Adjust source "test" start No
? Import "test" from a different JetStream domain Yes
? test Source foreign JetStream domain name leaf
? test Source foreign JetStream domain delivery prefix
Stream aggregate-test-leaf was created
```

Information for Stream aggregate-test-leaf created 2021-06-28T14:02:36-04:00

Configuration:

```
Acknowledgements: true
  Retention: File - Limits
    Replicas: 1
  Discard Policy: Old
Duplicate Window: 2m0s
Maximum Messages: unlimited
  Maximum Bytes: unlimited
    Maximum Age: 0.00s
Maximum Message Size: unlimited
Maximum Consumers: unlimited
  Sources: test, API Prefix: $JS.leaf.API, Delivery Prefix:
```

State:

```
Messages: 0
  Bytes: 0 B
  FirstSeq: 0
    LastSeq: 0
Active Consumers: 0
```

`source` as well as `mirror` take a copy of the messages. Once copied, accessing the data is independent of the leaf node connection being online. Copying this way also avoids having to run a dedicated program of your own. This is the recommended way to exchange persistent data across domains.

```
nats --server nats://acc:acc@localhost:4222 stream report --js-domain hub
```

Obtaining Stream stats

Stream Report							
Stream	Storage	Consumers	Messages	Bytes	Lost	Deleted	Replicas
backup-test-leaf	File	0	1	45 B	0	0	
test	File	0	1	45 B	0	0	
aggregate-test-leaf	File	0	1	98 B	0	0	

Replication Report						
Stream	Kind	API Prefix	Source Stream	Active	Lag	Error
backup-test-leaf	Mirror	\$JS.leaf.API	test	0.21s	0	
aggregate-test-leaf	Source	\$JS.leaf.API	test	1.23s	0	

Cross account & domain import

All of the above happened in the same account. To share domain access across accounts the `account.conf` from above needs to be modified and the server restarted or reloaded. This example exports the consumer and `FC` API as well as a delivery subject which is used by the internal push consumer created by `source` and `mirror`.

In support of another example on how to share a durable pull consumer for client access across domains and accounts, the `NEXT` and `ACK` API are exported as well.

🔧 *Known issue:* Currently, across accounts, push consumer are not supported.

On import, the JetStream API prefix `$JS.hub.API` is renamed to `JS.test@hub.API`. This is to, once more, disambiguate which JetStream a client in the importing account might want to interact with. When using domains, the general recommendation is to export the domain specific API `$JS.<domain>.API` as this allows you to pin the export to a particular domain.

Furthermore, the delivery subject is extended on import. This is to allow for easier export into multiple accounts.

This example also exports the absolute minimum necessary. It is possible to give access to the entire consumer API `$JS.hub.API.CONSUMER.>` or the entire API in a domain `$JS.hub.API.>` or the entire API `$JS.API.>` wherever the importing client connects.

```
accounts {
  SYS: {
    users: [{user: admin, password: admin}]
  },
  ACC: {
```

```

users: [{user: acc, password: acc}],
jetstream: enabled
exports: [
    # minimum export needed to allow source/mirror to create a consumer on the fly
    {service: "$JS.hub.API.CONSUMER.CREATE.*", response_type: "stream"}
    # minimum export needed for push consumer. This includes source and mirror!
    {stream: "deliver.acc.hub.>"}
    # minimum export needed for durable pull consumer `dur` in stream `aggregate-test-leaf`. (clients only - source and mirror do not use this)
    {service: "$JS.hub.API.CONSUMER.MSG.NEXT.aggregate-test-leaf.dur", response_type: "stream"}
    # minimum export needed to ack messages for durable consumer `dur` in stream `aggregate-test-leaf`. (clients only - source and mirror do not use this)
    {service: "$JS.ACK.aggregate-test-leaf.dur.>"}
    # minimum export needed for flow control of source/mirror
    {service: "$JS.FC.aggregate-test-leaf.dur.>"}
]
}
IMPORT_MIRROR: {
    users: [{user: import_mirror, password: import_mirror}],
    jetstream: enabled
    imports: [
        {service: {account: ACC, subject: "$JS.hub.API.CONSUMER.CREATE.*"}, to: "JS.acc@hub.API.CONSUMER.CREATE.*" }
        {service: {account: ACC, subject: "$JS.FC.aggregate-test-leaf.dur.>"}}
        {stream: {account: ACC, subject: deliver.acc.hub.import_mirror.>}}
    ]
}
# As of now, cross account, only pull consumer are supported.
IMPORT_CLIENT: {
    users: [{user: import_client, password: import_client}],
    jetstream: enabled
    imports: [
        {service: {account: ACC, subject: "$JS.hub.API.CONSUMER.MSG.NEXT.aggregate-test-leaf.dur"}, to: "JS.acc@hub.API.CONSUMER.MSG.NEXT.aggregate-test-leaf.dur" }
        {service: {account: ACC, subject: "$JS.ACK.aggregate-test-leaf.dur.>"}}
    ]
}
}
system_account: SYS

```

Copying via `source` and `mirror`

Once the servers have been restarted or reloaded, a `mirror` can be created as follows (same applies to `source`): On import from a different account the renamed prefix `JS.acc@hub.API` is provided. In addition, the delivery subject name is extended to also include the importing domain and stream. This makes it unique to that particular import. If every delivery prefix follows the pattern `<static type>.<exporting account>.<exporting domain>.<importing account>.<importing domain>.<importing stream name>` overlaps caused by multiple imports are avoided.

```
nats --server nats://import_mirror:import_mirror@localhost:4222 stream add --js-domain hub --mirror aggregate-test-leaf
```

```
? Stream Name aggregate-test-leaf-from-acc
? Storage backend file
? Retention Policy Limits
? Discard Policy Old
? Stream Messages Limit -1
? Message size limit -1
? Maximum message age limit -1
? Maximum individual message size -1
? Replicas 1
? Adjust mirror start No
? Import mirror from a different JetStream domain No
? Import mirror from a different account Yes
? Foreign account API prefix JS.acc@hub.API
? Foreign account delivery prefix deliver.acc.hub.import_mirror.hub.aggregate-test-leaf-from-acc
Stream aggregate-test-leaf-from-acc was created
```

Information for Stream aggregate-test-leaf-from-acc created 2021-06-28T16:59:15-04:00

Configuration:

```
Acknowledgements: true
  Retention: File - Limits
    Replicas: 1
  Discard Policy: Old
Duplicate Window: 2m0s
Maximum Messages: unlimited
  Maximum Bytes: unlimited
    Maximum Age: 0.00s
```

```
Mirror: aggregate-test-leaf, API Prefix: JS.acc@hub.API, Delivery Prefix:
deliver.acc.hub.import_mirror.hub.aggregate-test-leaf-from-acc
```

State:

```

    Messages: 0
      Bytes: 0 B
    FirstSeq: 0
      LastSeq: 0
Active Consumers: 0

```

A subsequent check shows that the one message stored in the stream aggregate in account `ACC` got copied to the new stream in the account `IMPORTER`.

```
nats --server nats://import_mirror:import_mirror@localhost:4222 stream report --js-domain hub
```

Obtaining Stream stats

```

┌
└─┬──────────┘
    |                               Stream
Report ───────────────────────────|
┌──────────────────────────────────┴──────────────────────────────────┐
└─┬──────────┘
    | Stream                    | Storage | Consumers | Messages | Bytes | Lost | Deleted |
Replicas ───────────────────────|──────────────────────────────────┐
┌──────────────────────────────────┴──────────────────────────────────┐
└─┬──────────┘
    | aggregate-test-leaf-from-acc | File   | 0       | 1       | 98 B  | 0      | 0
    |                             |
┌──────────────────────────────────┴──────────────────────────────────┐
└─┬──────────┘

```

Report | Replication

Stream	Kind	API Prefix	Source Stream	Active	Lag
Error					
aggregate-test-leaf-from-acc	Mirror	JS.acc@hub.API	aggregate-test-leaf	0.59s	0

Direct access of a durable pull consumer

The modified `accounts.conf` also includes a separate import for an existing pull consumer. Let's create a consumer by the name `dur` in the stream `aggregate-test-leaf` in the account `acc`.

```
nats --server nats://acc:acc@localhost:4222 consumer add --js-domain hub
```

```
? Consumer name dur
? Delivery target (empty for Pull Consumers)
? Start policy (all, new, last, lh, msg sequence) all
? Replay policy instant
? Filter Stream by subject (blank for all)
? Maximum Allowed Deliveries -1
? Maximum Acknowledgements Pending 0
? Select a Stream aggregate-test-leaf
Information for Consumer aggregate-test-leaf > dur created 2021-06-28T17:16:51-04:00
```

Configuration:

```

Durable Name: dur
  Pull Mode: true
  Deliver All: true
  Ack Policy: Explicit
    Ack Wait: 30s
  Replay Policy: Instant
  Max Ack Pending: 20,000
  Max Waiting Pulls: 512

```

State:

Last Delivered Message: Consumer sequence: 0 Stream sequence: 0
Acknowledgment floor: Consumer sequence: 0 Stream sequence: 0
Outstanding Acks: 0 out of maximum 20000
Redelivered Messages: 0
Unprocessed Messages: 1
Waiting Pulls: 0 of maximum 512

```
nats --server nats://acc:acc@localhost:4222 stream report --js-domain hub
```

Obtaining Stream stats

Stream Report							
Stream	Storage	Consumers	Messages	Bytes	Lost	Deleted	Replicas
backup-test-leaf	File	0	1	45 B	0	0	
test	File	0	1	45 B	0	0	
aggregate-test-leaf	File	1	1	98 B	0	0	

Replication Report						
Stream	Kind	API Prefix	Source Stream	Active	Lag	Error
backup-test-leaf	Mirror	\$JS.leaf.API	test	1.85s	0	
aggregate-test-leaf	Source	\$JS.leaf.API	test	1.85s	0	

```
nats --server nats://acc:acc@localhost:4222 consumer report --js-domain hub
```

Output

? Select a Stream aggregate-test-leaf

Consumer report for aggregate-test-leaf with 1 consumers	

Consumer	Mode	Ack Policy	Ack Wait	Ack Pending	Redelivered	Unprocessed	Ack	
Floor	Cluster							
dur	Pull	Explicit	30.00s	0	0	1 / 100%		
0	hub-server*							

To retrieve the messages stored in the domain `hub` using `nats` while connected to the leaf node, provide the correct stream and durable name as well as the API prefix `JS.acc@hub.API`

```
nats --server nats://import_client:import_client@localhost:4111 consumer next aggregate-test-leaf dur --js-api-prefix JS.acc@hub.API
```

[17:44:16] subj: test / tries: 1 / cons seq: 1 / str seq: 1 / pending: 0

Headers:

Nats-Stream-Source: test:mSx7q4yJ 1

Data:

hello world

Acknowledged message

```
nats --server nats://acc:acc@localhost:4222 consumer report --js-domain hub
```

? Select a Stream aggregate-test-leaf

Consumer report for aggregate-test-leaf with 1 consumers

Consumer	Mode	Ack Policy	Ack Wait	Ack Pending	Redelivered	Unprocessed	Ack	


```

func main() {
    nc, err := nats.Connect(os.Args[1], nats.Name("JS test"))
    defer nc.Close()
    if err != nil {
        fmt.Printf("nats connect: %v\n", err)
        return
    }
    js, err := nc.JetStream(nats.APIPrefix("JS.acc@hub.API"))
    if err != nil {
        fmt.Printf("JetStream: %v\n", err)
        if js == nil {
            return
        }
    }
    s, err := js.PullSubscribe("", "dur", nats.Bind("aggregate-test-leaf", "dur"))
    if err != nil {
        fmt.Printf("PullSubscribe: %v\n", err)
        return
    }

    shutdown := make(chan os.Signal, 1)
    signal.Notify(shutdown, os.Interrupt, syscall.SIGTERM)

    fmt.Printf("starting\n")
    for {
        select {
        case <-shutdown:
            return
        default:
            if m, err := s.Fetch(1, nats.MaxWait(time.Second)); err != nil {
                fmt.Println(err)
            } else {

                if meta, err := m[0].Metadata(); err == nil {
                    fmt.Printf("%+v\n", meta)
                }
                fmt.Println(string(m[0].Data))

                if err := m[0].Ack(); err != nil {
                    fmt.Printf("ack error: %+v\n", err)
                }
            }
        }
    }
}

```

```
    }
  }
}
}
```

A push subscriber will need a similar setup. It will require the `ACK` subject. However, instead of exporting/importing the `NEXT` subject, the delivery subject shown for source/mirror needs to be used.

See Also

[JetStream Leaf Nodes demo](#) and its companion [video recording](#).

📄	#3	
📄	cocobook-sync	📄 2021-11-22 23:51:02 UTC
📄	cocobook-sync	📄 2024-08-26 16:27:24 UTC