

System Events



- [Available Events and Services](#)
 - [System Account](#)
 - [Local Configuration](#)
-

NATS servers leverage [Accounts](#) support and generate events such as:

- account connect/disconnect
- authentication errors
- server shutdown
- server stat summary

In addition the server supports a limited number of requests that can be used to query for account connections, server stat summaries, and pinging servers in the cluster.

These events are enabled by configuring `system_account` and [subscribing/requesting](#) using a *system account* user.

[Accounts](#) are used so that subscriptions from your applications, say `>`, do not receive system events and vice versa. Using accounts requires either:

- [Configuring authentication locally](#) and listing one of the accounts in `system_account`
- Or by using decentralized authentication and authorization via [jwt](#) as shown in this [Tutorial](#). In this case `system_account` contains the account public key.

N.B. The default global account `$G` does not publish advisories.

Available Events and Services

System Account

The system account publishes messages under well known subject patterns.

Server initiated events:

- `$SYS.ACCOUNT.<id>.CONNECT` (client connects)
- `$SYS.ACCOUNT.<id>.DISCONNECT` (client disconnects)
- `$SYS.ACCOUNT.<id>.SERVER.CONNS` (connections for an account changed)
- `$SYS.SERVER.<id>.CLIENT.AUTH.ERR` (authentication error)
- `$SYS.SERVER.<id>.STATSZ` (stats summary)

In addition other tools with system account privileges, can initiate requests (Examples can be found [here](#)):

- `$SYS.REQ.SERVER.<id>.STATSZ` (request server stat summary)
- `$SYS.REQ.SERVER.PING` (discover servers - will return multiple messages)

[Monitoring endpoints](#) as listed in the table below are accessible as system services using the following subject pattern:

- `$SYS.REQ.SERVER.<id>.<endpoint-name>` (request server monitoring endpoint corresponding to endpoint name.)
- `$SYS.REQ.SERVER.PING.<endpoint-name>` (from all server, request server monitoring endpoint corresponding to endpoint name - will return multiple messages)

Endpoint	Endpoint Name
General Server Information	<code>VARZ</code>
Connections	<code>CONNZ</code>
Routing	<code>ROUTEZ</code>
Gateways	<code>GATEWAYZ</code>
Leaf Nodes	<code>LEAFZ</code>
Subscription Routing	<code>SUBSZ</code>
JetStream	<code>JSZ</code>
Accounts	<code>ACCOUNTZ</code>
Health	<code>HEALTHZ</code>

- `"$SYS.REQ.ACCOUNT.<account-id>.<endpoint-name>` (from all server, request account specific monitoring endpoint corresponding to account id and endpoint name - will return multiple messages)

Endpoint	Endpoint Name
Connections	<code>CONNZ</code>

Endpoint	Endpoint Name
Leaf Nodes	LEAFZ
Subscription Routing	SUBSZ
JetStream	JSZ
Account	INFO

Servers like `nats-account-server` publish system account messages when a claim is updated, the `nats-server` listens for them, and updates its account information accordingly:

- `$SYS.ACCOUNT.<id>.CLAIMS.UPDATE`

With these few messages you can build useful monitoring tools:

- health/load of your servers
- client connects/disconnects
- account connections
- authentication errors

Local Configuration

To make use of System events, just using accounts, your configuration can look like this:

```
accounts: {
  USERS: {
    users: [
      {user: a, password: a}
    ]
  },
  SYS: {
    users: [
      {user: admin, password: changeit}
    ]
  },
}
system_account: SYS
```

Please note that applications now have to authenticate such that a connection can be associated with an account. In this example username and password were chosen for simplicity of the demonstration. Subscribe to all system events like this `nats sub -s`
`nats://admin:changeit@localhost:4222 ">"` and observe what happens when you do something like

`nats pub -s "nats://a:a@localhost:4222" foo bar`. Examples on how to use system services can be found [here](#).

📄 #3

📄 cocobook-sync 📄 2026-10-28 11:14:25 UTC

📄 cocobook-sync 📄 2026-10-29 01:42:39 UTC