

Listening for Reconnect Events

Because reconnect is primarily under the covers, many libraries provide an event listener you can use to be notified of reconnect events. This event can be especially important for applications sending a lot of messages.

Go

```
// Connection event handlers are invoked asynchronously
// and the state of the connection may have changed when
// the callback is invoked.
nc, err := nats.Connect("demo.nats.io",
    nats.DisconnectErrHandler(func(nc *nats.Conn, err error) {
        // handle disconnect error event
    }),
    nats.ReconnectHandler(func(nc *nats.Conn) {
        // handle reconnect event
    }))
if err != nil {
    log.Fatal(err)
}
defer nc.Close()

// Do something with the connection
```

Java

```
class MyConnectionListener implements ConnectionListener {
    public void connectionEvent(Connection conn, Events event) {
        switch (event) {
            case CONNECTED:
                // The connection has successfully completed the handshake with the nats-
server.

                break;
            case CLOSED:
```

```

        // The connection is permanently closed, either by manual action or failed
reconnects
        break;
    case DISCONNECTED:
        // The connection lost its connection, but may try to reconnect if configured
to
        break;
    case RECONNECTED:
        // The connection was connected, lost its connection and successfully
reconnected.
        break;
    case RESUBSCRIBED:
        // The connection was reconnected and the server has been notified of all
subscriptions.
        break;
    case DISCOVERED_SERVERS:
        // The connection was made aware of new servers from the current server
connection.
        break;
    case LAME_DUCK:
        // connected server is coming down soon, might want to prepare for it
        break;
    }
}
}

MyConnectionListener listener = new MyConnectionListener();

Options options = new Options.Builder()
    .server("nats://demo.nats.io:4222")
    .connectionListener(listener)
    .build();
Connection nc = Nats.connect(options);

// Do something with the connection

nc.close();

```

JavaScript

```

const nc = await connect({
  maxReconnectAttempts: 10,
  servers: ["demo.nats.io"],
});

(async () => {
  for await (const s of nc.status()) {
    switch (s.type) {
      case Events.Reconnect:
        t.log(`client reconnected - ${s.data}`);
        break;
      default:
    }
  }
})();
});

```

Python

```

nc = NATS()

async def disconnected_cb():
    print("Got disconnected!")

async def reconnected_cb():
    # See who we are connected to on reconnect.
    print("Got reconnected to {url}".format(url=nc.connected_url.netloc))

await nc.connect(
    servers=["nats://demo.nats.io:4222"],
    reconnect_time_wait=10,
    reconnected_cb=reconnected_cb,
    disconnected_cb=disconnected_cb,
)

# Do something with the connection.

```

C#

```
// dotnet add package NATS.Net
using NATS.Net;

await using var client = new NatsClient();

client.Connection.ConnectionDisconnected += async (sender, args) =>
{
    Console.WriteLine($"Disconnected: {args.Message}");
};

client.Connection.ConnectionOpened += async (sender, args) =>
{
    Console.WriteLine($"Connected: {args.Message}");
};

client.Connection.ReconnectFailed += async (sender, args) =>
{
    Console.WriteLine($"Reconnect Failed: {args.Message}");
};

await client.ConnectAsync();
```

Ruby

```
require 'nats/client'

NATS.start(servers: ["nats://127.0.0.1:1222", "nats://127.0.0.1:1223",
"nats://127.0.0.1:1224"]) do |nc|
    # Do something with the connection
    nc.on_reconnect do
        puts "Got reconnected to #{nc.connected_server}"
    end

    nc.on_disconnect do |reason|
        puts "Got disconnected! #{reason}"
    end
end
```

C

```

static void
disconnectedCB(natsConnection *conn, void *closure)
{
    // Handle disconnect error event
}

static void
reconnectedCB(natsConnection *conn, void *closure)
{
    // Handle reconnect event
}

(...)

natsConnection      *conn      = NULL;
natsOptions          *opts      = NULL;
natsStatus           s          = NATS_OK;

s = natsOptions_Create(&opts);

// Connection event handlers are invoked asynchronously
// and the state of the connection may have changed when
// the callback is invoked.
if (s == NATS_OK)
    s = natsOptions_SetDisconnectedCB(opts, disconnectedCB, NULL);
if (s == NATS_OK)
    s = natsOptions_SetReconnectedCB(opts, reconnectedCB, NULL);

if (s == NATS_OK)
    s = natsConnection_Connect(&conn, opts);

(...)

// Destroy objects that were created
natsConnection_Destroy(conn);
natsOptions_Destroy(opts);

```

📄 #1

📄 cocobook-sync 📄 2026-10-28 11:12:30 UTC

📄 cocobook-sync 📄 2026-10-28 11:12:30 UTC