

Listen for Connection Events



- [Listen for New Servers](#)
 - [Listen for Errors](#)
-

While the connection status is interesting, it is perhaps more interesting to know when the status changes. Most, if not all, of the NATS client libraries provide a way to listen for events related to the connection and its status.

The actual API for these listeners is language dependent, but the following examples show a few of the more common use cases. See the API documentation for the client library you are using for more specific instructions.

Connection events may include the connection being closed, disconnected or reconnected. Reconnecting involves a disconnect and connect, but depending on the library implementation may also include multiple disconnects as the client tries to find a server, or the server is rebooted.

Go

```
// There is not a single listener for connection events in the NATS Go Client.
// Instead, you can set individual event handlers using:
nc, err := nats.Connect("demo.nats.io",
    nats.DisconnectErrHandler(func(_ *nats.Conn, err error) {
        log.Printf("client disconnected: %v", err)
    }),
    nats.ReconnectHandler(func(_ *nats.Conn) {
        log.Printf("client reconnected")
    }),
    nats.ClosedHandler(func(_ *nats.Conn) {
        log.Printf("client closed")
    }))
if err != nil {
    log.Fatal(err)
}
defer nc.Close()
```

```
DisconnectHandler(cb ConnHandler)
ReconnectHandler(cb ConnHandler)
ClosedHandler(cb ConnHandler)
DiscoveredServersHandler(cb ConnHandler)
ErrorHandler(cb ErrHandler)
```

Java

```
class MyConnectionListener implements ConnectionListener {
    public void connectionEvent(Connection natsConnection, Events event) {
        System.out.println("Connection event - " + event);
    }
}

public class SetConnectionListener {
    public static void main(String[] args) {
        try {
            Options options = new Options.Builder()
                .server("nats://demo.nats.io:4222")
                .connectionListener(new MyConnectionListener()) // Set the listener
                .build();
            Connection nc = Nats.connect(options);

            // Do something with the connection

            nc.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JavaScript

```
const nc = await connect({ servers: ["demo.nats.io"] });
nc.closed().then(() => {
    t.log("the connection closed!");
});

(async () => {
```

```

for await (const s of nc.status()) {
  switch (s.type) {
    case Events.Disconnect:
      t.log(`client disconnected - ${s.data}`);
      break;
    case Events.LDM:
      t.log("client has been requested to reconnect");
      break;
    case Events.Update:
      t.log(`client received a cluster update - ${s.data}`);
      break;
    case Events.Reconnect:
      t.log(`client reconnected - ${s.data}`);
      break;
    case Events.Error:
      t.log("client got a permissions error");
      break;
    case DebugEvents.Reconnecting:
      t.log("client is attempting to reconnect");
      break;
    case DebugEvents.StaleConnection:
      t.log("client has a stale connection");
      break;
    default:
      t.log(`got an unknown status ${s.type}`);
  }
}
})).then();

```

Python

```

# Asyncio NATS client can be defined a number of event callbacks
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))

async def error_cb(e):

```

```

    print("There was an error: {}".format(e))

async def closed_cb():
    print("Connection is closed")

# Setup callbacks to be notified on disconnects and reconnects
options["disconnected_cb"] = disconnected_cb
options["reconnected_cb"] = reconnected_cb

# Setup callbacks to be notified when there is an error
# or connection is closed.
options["error_cb"] = error_cb
options["closed_cb"] = closed_cb

await nc.connect(**options)

```

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

```
# There is not a single listener for connection events in the Ruby NATS Client.
```

```
# Instead, you can set individual event handlers using:
```

```
NATS.on_disconnect do  
end
```

```
NATS.on_reconnect do  
end
```

```
NATS.on_close do  
end
```

```
NATS.on_error do  
end
```

C

```
static void  
disconnectedCB(natsConnection *conn, void *closure)  
{  
    // Do something  
    printf("Connection disconnected\n");  
}
```

```
static void  
reconnectedCB(natsConnection *conn, void *closure)  
{  
    // Do something  
    printf("Connection reconnected\n");  
}
```

```
static void  
closedCB(natsConnection *conn, void *closure)  
{  
    // Do something  
    printf("Connection closed\n");  
}
```

```
(...)
```

```

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

s = natsOptions_Create(&opts);
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 = natsOptions_SetClosedCB(opts, closedCB, NULL);
if (s == NATS_OK)
    s = natsConnection_Connect(&conn, opts);

(...)

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

```

Listen for New Servers

When working with a cluster, servers may be added or changed. Some of the clients allow you to listen for this notification:

Go

```

// Be notified if a new server joins the cluster.
// Print all the known servers and the only the ones that were discovered.
nc, err := nats.Connect("demo.nats.io",
    nats.DiscoveredServersHandler(func(nc *nats.Conn) {
        log.Printf("Known servers: %v\n", nc.Servers())
        log.Printf("Discovered servers: %v\n", nc.DiscoveredServers())
    }))
if err != nil {
    log.Fatal(err)
}
defer nc.Close()

```

```
// Do something with the connection
```

Java

```
class ServersAddedListener implements ConnectionListener {
    public void connectionEvent(Connection nc, Events event) {
        if (event == Events.DISCOVERED_SERVERS) {
            for (String server : nc.getServers()) {
                System.out.println("Known server: "+server);
            }
        }
    }
}

public class ListenForNewServers {
    public static void main(String[] args) {

        try {
            Options options = new Options.Builder().
                server("nats://demo.nats.io:4222").
                connectionListener(new ServersAddedListener()). // Set
the listener
                build();

            Connection nc = Nats.connect(options);

            // Do something with the connection

            nc.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JavaScript

```
const nc = await connect({ servers: ["demo.nats.io:4222"] });
(async () => {
    for await (const s of nc.status()) {
```

```

switch (s.type) {
  case Status.Update:
    t.log(`servers added - ${s.data.added}`);
    t.log(`servers deleted - ${s.data.deleted}`);
    break;
  default:
}
}
}().then();

```

Python

```
# Asyncio NATS client does not support discovered servers handler right now
```

C#

```
// NATS .NET client does not support discovered servers handler right now
```

Ruby

```
# The Ruby NATS client does not support discovered servers handler right now
```

C

```

static void
discoveredServersCB(natsConnection *conn, void *closure)
{
    natsStatus s = NATS_OK;
    char **servers = NULL;
    int count = 0;

    s = natsConnection_GetDiscoveredServers(conn, &servers, &count);
    if (s == NATS_OK)
    {
        int i;

        // Do something...
        for (i=0; i<count; i++)
            printf("Discovered server: %s\n", servers[i]);
    }
}

```

```

        // Free allocated memory
        for (i=0; i<count; i++)
            free(servers[i]);
        free(servers);
    }
}

(...)

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

s = natsOptions_Create(&opts);
if (s == NATS_OK)
    s = natsOptions_SetDiscoveredServersCB(opts, discoveredServersCB, NULL);
if (s == NATS_OK)
    s = natsConnection_Connect(&conn, opts);

(...)

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

```

Listen for Errors

The client library may separate server-to-client errors from events. Many server events are not handled by application code and result in the connection being closed. Listening for the errors can be very useful for debugging problems.

Go

```

// Set the callback that will be invoked when an asynchronous error occurs.
nc, err := nats.Connect("demo.nats.io",
    nats.ErrorHandler(func(_ *nats.Conn, _ *nats.Subscription, err error) {
        log.Printf("Error: %v", err)
    })))

```

```

if err != nil {
    log.Fatal(err)
}
defer nc.Close()

// Do something with the connection

```

Java

```

class MyErrorListener implements ErrorListener {
    public void errorOccurred(Connection conn, String error)
    {
        System.out.println("The server notified the client with: "+error);
    }

    public void exceptionOccurred(Connection conn, Exception exp) {
        System.out.println("The connection handled an exception: "+exp.getLocalizedMessage());
    }

    public void slowConsumerDetected(Connection conn, Consumer consumer) {
        System.out.println("A slow consumer was detected.");
    }
}

public class SetErrorListener {
    public static void main(String[] args) {

        try {
            Options options = new Options.Builder().
                server("nats://demo.nats.io:4222").
                errorListener(new MyErrorListener()). // Set the
listener
                build();

            Connection nc = Nats.connect(options);

            // Do something with the connection

            nc.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
  }
}

```

JavaScript

```

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

// if the client gets closed with an error you can trap that
// condition in the closed handler like this:
nc.closed().then((err) => {
  if (err) {
    t.log(`the connection closed with an error ${err.message}`);
  } else {
    t.log(`the connection closed.`);
  }
});

// if you have a status listener, it will too get notified
(async () => {
  for await (const s of nc.status()) {
    switch (s.type) {
      case Status.Error:
        // typically if you get this the nats connection will close
        t.log("client got an async error from the server");
        break;
      default:
        t.log(`got an unknown status ${s.type}`);
    }
  }
})();

```

Python

```

nc = NATS()

async def error_cb(e):
    print("Error: ", e)

await nc.connect(

```

```
servers=["nats://demo.nats.io:4222"],
reconnect_time_wait=10,
error_cb=error_cb,
)
```

```
# Do something with the connection.
```

C#

```
// dotnet add package NATS.Net
using Microsoft.Extensions.Logging;
using NATS.Client.Core;
using NATS.Net;

// NATS .NET client does not support error handler right now
// instead, you can use the logger since server errors are logged
// with the error level and eventId 1005 (Protocol Log Event).
await using var client = new NatsClient(new NatsOpts
{
    LoggerFactory = LoggerFactory.Create(builder => builder.AddConsole()),
});
```

Ruby

```
require 'nats/client'

NATS.start(servers:["nats://demo.nats.io:4222"]) do |nc|
  nc.on_error do |e|
    puts "Error: #{e}"
  end

  nc.close
end
```

C

```
static void
errorCB(natsConnection *conn, natsSubscription *sub, natsStatus s, void *closure)
{
    // Do something
    printf("Error: %d - %s\n", s, natsStatus_GetText(s));
}
```

```

}

(...)

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

s = natsOptions_Create(&opts);
if (s == NATS_OK)
    s = natsOptions_SetErrorHandler(opts, errorCallback, 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 11:12:34 UTC

📄 cocobook-sync 📄 2026-10-28 11:12:34 UTC