

TLS Authentication



- [Validating a Client Certificate](#)
 - [Mapping Client Certificates To A User](#)
 - [TLS Timeout](#)
-

The server can require TLS certificates from a client. When needed, you can use the certificates to:

- Validate the client certificate matches a known or trusted CA
- Extract information from a trusted certificate to provide authentication

“ Note: To simplify the common scenario of maintainers looking at the monitoring endpoint, `verify` and `verify_and_map` do not apply to the monitoring port.

The examples in the following sections make use of the certificates you [generated](#) locally.

Validating a Client Certificate

The server can verify a client certificate using a CA certificate. To require verification, add the option `verify` to the TLS configuration section as follows:

```
tls {  
  cert_file: "server-cert.pem"  
  key_file:  "server-key.pem"  
  ca_file:   "rootCA.pem"  
  verify:    true  
}
```

Or via the command line:

```
nats-server --tlsverify --tlscert=server-cert.pem --tlskey=server-key.pem --  
tlscacert=rootCA.pem
```

This option verifies the client's certificate is signed by the CA specified in the `ca_file` option. When `ca_file` is not present it will default to CAs in the system trust store. It also makes sure that the client provides a certificate with the extended key usage `TLS Web Client Authentication`.

Mapping Client Certificates To A User

In addition to verifying that a specified CA issued a client certificate, you can use information encoded in the certificate to authenticate a client. The client wouldn't have to provide or track usernames or passwords.

To have TLS Mutual Authentication map certificate attributes to the user's identity use `verify_and_map` as shown as follows:

```
tls {  
  cert_file: "server-cert.pem"  
  key_file:  "server-key.pem"  
  ca_file:   "rootCA.pem"  
  # Require a client certificate and map user id from certificate  
  verify_and_map: true  
}
```

“ Note that `verify` was changed to `verify_and_map`.

When present, the server will check if a Subject Alternative Name (SAN) maps to a user. It will search all email addresses first, then all DNS names. If no user could be found, it will try the certificate subject.

“ Note: This mechanism will pick the user it finds first. There is no configuration to restrict this.

```
openssl x509 -noout -text -in client-cert.pem
```

Certificate:

...

X509v3 extensions:

X509v3 Subject Alternative Name:

DNS:localhost, IP Address:0:0:0:0:0:0:0:1, email:email@localhost

X509v3 Extended Key Usage:

TLS Web Client Authentication

...

The configuration to authorize this user would be as follow:

```
authorization {
  users = [
    {user: "email@localhost"}
  ]
}
```

Use the [RFC 2253 Distinguished Names](#) syntax to specify a user corresponding to the certificate subject:

```
openssl x509 -noout -text -in client-cert.pem
```

Certificate:

Data:

...

Subject: 0=mkcert development certificate, OU=testuser@MacBook-Pro.local (Test User)

...

“ Note that for this example to work you will have to modify the user to match what is in your certificates subject. In doing so, watch out for the order of attributes!

The configuration to authorize this user would be as follows:

```
authorization {
  users = [
    {user: "OU=testuser@MacBook-Pro.local (Test User),0=mkcert development certificate"}
  ]
}
```

TLS Timeout

[TLS timeout](#) is described here.

[] [] #1	
[] cocobook-sync [] []	2026-10-28 11:14:02 UTC
[] cocobook-sync [] []	2026-10-28 11:14:02 UTC