

Decentralized JWT Authentication/Authorization



- [JSON Web Tokens](#)
 - [NKey Roles](#)
 - [The Authentication Process](#)
 - [The Authorization Process](#)
 - [JWTs and Privacy](#)
 - [Decentralized Authentication and Authorization - Configuration and nsc](#)
 - [Managing JWT authentication](#)
-

With other authentication mechanisms, configuration for identifying a user and [Account](#), is in the server configuration file. JWT authentication leverages [JSON Web Tokens \(JWT\)](#) to describe the various entities supported. When a client connects, servers verify the authenticity of the request using [NKeys](#), download account information and validate a trust chain. Users are not directly tracked by the server, but rather verified as and belonging to an [Account](#). This enables the management of users, without requiring server configuration updates.

Effectively, JWTs improve accounts and provide for a **distributed configuration paradigm**. Previously each user (or client) needed to be known and authorized a priori in the server's configuration requiring an administrator to modify and update server configurations. These chores are eliminated. User creation can even be performed by different entities altogether.

“ Note: This scheme improves [accounts](#). Functionalities like [isolation](#) or defining [exports/imports](#) between accounts remain! It moves configuration of accounts, exports/imports or users and their permissions away from the server into several trusted [JSON Web Token \(JWT\)](#) that are managed separately, therefore removing the need to configure these entities in each and every server. It furthermore adds functionalities like expiration and revocation fore decentralized account

JSON Web Tokens

[JSON Web Tokens \(JWT\)](#) are an open and industry standard [RFC7519](#) method for representing claims securely between two parties.

Claims are a fancy way of asserting information on a *subject*. In this context, a *subject* is the entity being described (not a messaging subject). Standard JWT claims are typically digitally signed and verified.

NATS further restricts JWTs by requiring that JWTs be:

- Digitally signed *always* and only using [Ed25519](#).
- NATS adopts the convention that all *Issuer* and *Subject* fields in a JWT claim must be a public [NKEY](#).
- *Issuer* and *Subject* must match specific roles depending on the claim [NKeys](#).

NKey Roles

[NKeys](#) Roles are:

- Operators
- Accounts
- Users

Roles are hierarchical and form a chain of trust. Operators issue Accounts which in turn issue Users. Servers trust specific Operators. If an account is issued by an operator that is trusted, account users are trusted.

The Authentication Process

When a *User* connects to a server, it presents a JWT issued by its *Account*. The user proves its identity by signing a server-issued cryptographic challenge with its private key. The signature verification validates that the signature is attributable to the user's public key. Next, the server retrieves the associated account JWT that issued the user. It verifies the *User* issuer matches the referenced account. Finally, the server checks that a trusted *Operator* - one the server is

configured with - issued the *Account*, completing the trust chain verification.

The Authorization Process

From an authorization point of view, the account provides information on messaging subjects that are imported from other accounts (including any ancillary related authorization) as well as messaging subjects exported to other accounts. Accounts can also bear limits, such as the maximum number of connections they may have. A user JWT can express restrictions on the messaging subjects to which it can publish or subscribe.

When a new user is added to an account, the account configuration need not change, as each user can and should have its own user JWT that can be verified by simply resolving its parent account.

JWTs and Privacy

One crucial detail to keep in mind is that while in other systems JWTs are used as sessions or proof of authentication, NATS JWTs are only used as configuration describing:

- the public ID of the entity
- the public ID of the entity that issued it
- capabilities of the entity

Authentication is a public key cryptographic process — a client signs a nonce proving identity while the trust chain and configuration provides the authorization.

The server is never aware of private keys but can verify that a signer or issuer indeed matches a specified or known public key.

Lastly, all NATS JWTs (Operators, Accounts, Users and others) are expected to be signed using the [Ed25519](#) algorithm. If they are not, they are rejected by the system.

Decentralized Authentication and Authorization - Configuration and nsc

There is very little to configure on the nats-server to enable operator JWT security, once the servers have been initially configured the authentication and authorization tasks are typically done by using the `nsc` administration tool locally and synchronizing with the account resolvers built into the nats-server.

Configuration is broken up into separate steps. Depending on organizational needs these are performed by the same or different entities.

Practically, JWT configuration is done using the `nsc` tool. It can be set up to issue [NKeys](#) and corresponding JWTs for all [nkey roles](#): Operator/Account/User ([Example usage](#)). Despite Account and User creation not happening in server configuration, this model is a centralized authentication and authorization setup.

Provided institutional trust, it is also possible to use nsc to import account or user public [NKeys](#) and issue corresponding JWTs. This way an operator can issue account JWTs and a separate entity can issue JWTs for user associated with it's account. Neither entity has to be aware of the other's private Nkey. This not only allows users to be configured some place other than servers, but also by different organizations altogether. Say administrators of a NATS installation controlling operators, issuing account JWTs to individual prod/dev teams managing their own user. This is a fully decentralized authorization setup!

With an Operator JWT in place, the server needs to be configured to trust it by specifying `operator`. Furthermore the server needs a way to obtain account JWTs. This done by either defaulting to the resolver specified in the operator jwt or by manually specifying the [resolver](#). Depending on your configuration an [account server](#) needs to be in place

“ It is possible to [mix](#) JWT and [NKEY/Account](#) based Authentication/Authorization.

Managing JWT authentication

A lot more information is available in the [In Depth Guide](#).

📄 #3

📄 cocobook-sync 📄 2026-10-28 11:14:08 UTC

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