

Account lookup using Resolver



- [NATS Based Resolver](#)
 - [Full](#)
 - [Cache](#)
 - [NATS-Based Resolver - Integration](#)
 - [Migrating account data](#)
 - [MEMORY](#)
 - [URL Resolver](#)
-

The `resolver` configuration option is used in conjunction with [NATS JWT Authentication](#) and [nsc](#). The `resolver` option specifies a URL where the nats-server can retrieve an account JWT. There are 3 resolver implementations:

- [NATS Based Resolver](#) which is the preferred option and should be your default selection
- `MEMORY` if you want to statically define the accounts in the server configuration
- `URL` if you want to build your own account service, typically in order to have some integration of NATS security with some external security system.

“ If the operator JWT specified in `operator` contains an account resolver URL, `resolver` only needs to be specified in order to overwrite that default.

NATS Based Resolver

The NATS based resolver is the preferred and easiest way to enable account lookup for the nats servers. It is built-in into `nats-server` and stores the account JWTs in a local (not shared) directory that the server has access to (i.e. you can't have more than one `nats-server`s using the same directory. All the servers in the cluster or super-cluster must be configured to use it, and they implement an 'eventually consistent' mechanism via NATS and the system account to synchronize

(or lookup) the account data between themselves.

In order to avoid having to store all account JWT on every `nats-server` (i.e. if you have a *lot* of accounts), this resolver has two sub types `full` and `cache`.

In this mode of operation administrators typically use the `nsc` CLI tool to create/manage the JWTs locally, and use `nsc push` to push new JWTs to the nats-servers' built-in resolvers, `nsc pull` to refresh their local copy of account JWTs, and `nsc revocations` to revoke them.

Full

The Full resolver means that the `nats-server` stores all JWTs and exchanges them in an eventually consistent way with other resolvers of the same type.

```
resolver: {
  type: full
  # Directory in which account jwt will be stored
  dir: './jwt'
  # In order to support jwt deletion, set to true
  # If the resolver type is full delete will rename the jwt.
  # This is to allow manual restoration in case of inadvertent deletion.
  # To restore a jwt, remove the added suffix .delete and restart or send a reload signal.
  # To free up storage you must manually delete files with the suffix .delete.
  allow_delete: false
  # Interval at which a nats-server with a nats based account resolver will compare
  # it's state with one random nats based account resolver in the cluster and if needed,
  # exchange jwt and converge on the same set of jwt.
  interval: "2m"
  # limit on the number of jwt stored, will reject new jwt once limit is hit.
  limit: 1000
}
```

This resolver type also supports `resolver_preload`. When present, JWTs are listed and stored in the resolver. There, they may be subject to updates. Restarts of the `nats-server` will hold on to these more recent versions.

Not every server in a cluster needs to be set to `full`. You need enough to still serve your workload adequately, while some servers are offline.

Cache

The Cache resolver means that the `nats-server` only stores a subset of the JWTs and evicts others based on an LRU scheme. The cache relies on (a) `full` NATS-based resolver(s) to retrieve accounts not present in the cache. A cache resolver does NOT accept account push messages from nsc and therefore is not suitable for stand-alone operation without a full resolver present.

```
resolver: {
  type: cache
  # Directory in which account jwt will be store
  dir: "./"
  # limit on the number of jwt stored, will evict old jwt once limit is hit.
  limit: 1000
  # How long to hold on to a jwt before discarding it.
  ttl: "2m"
}
```

NATS-Based Resolver - Integration

The NATS-based resolver utilizes the system account for lookup and upload of account JWTs. If your application requires tighter integration you can make use of these subjects for tighter integration.

To upload or update any generated account JWT without `nsc`, send it as a request to `$SYS.REQ.CLAIMS.UPDATE`. Each participating `full` NATS-based account resolver will respond with a message detailing success or failure.

To serve a requested account JWT yourself and essentially implement an account server, subscribe to `$SYS.REQ.ACCOUNT.*.CLAIMS.LOOKUP` and respond with the account JWT corresponding to the requested account id (wildcard).

Migrating account data

To migrate account data when you change from using the standalone (REST) account server to the built-in NATS account resolver (or between NATS environments, or account servers) you can use `nsc`:

1. Run `nsc pull` to make sure you have a copy of all the account data in the server in your local machine
2. Reconfigure your servers to use the nats resolver instead of the URL resolver
3. Modify the 'account server URL' setting in your operator to the nats URL from the old REST URL: i.e. just copy the nats URLs from the operator's 'service URLs' setting into the account server URLs. `nsc edit operator --account-jwt-server-url <nats://...>`
4. do `nsc push -A` to push your account data to the nats-servers using the built-in nats account resolver

You can also pass the account server URLs directly as a flag to the `nsc pull` and `nsc push` commands.

MEMORY

The `MEMORY` resolver is statically configured in the server's configuration file. You would use this mode if you would rather manage the account resolving 'by hand' through the `nat-server`'s configuration files. The memory resolver makes use of the `resolver_preload` directive, which specifies a map of public keys to account JWTs:

```
resolver: MEMORY
resolver_preload: {
  ACSU3Q6LTBLVGAQUONAGXJHVNWGSKKAUA7IY5TB4Z7PLEKSR506JTGR:
    eyJ0eXAiOiJqd3QiLCJhbGciOiJIU2I1NTU5In0.eyJqdGkiOiJPRFhJSVI2Wlg1Q1AzMlFJTTFczWFBENEtTSDYzUFNNSE
    ZHUkpaT05DR1RLVVBISlRLQ0JBW0IiwiaWF0IjoxNTU2NjU1Njk0LCJpc3MiOiJPRFdaSjJLQVBNZzZXT1dNUENKRjZCWTRR
    SVBMVFVJWTRKSUJMVTRLMT1ERzNHSElXQlZXQkhVWiIsIm5hbWUiOiJBIiwic3ViIjojQUQNTVTRNRkxUTEJWTEdBUVVPk
    FHWEpIVk5XR1NLS0FVQTDJWTVUQjRan1BMRUtuTjVpNkxUR1IiLCJ0eXB1IjojYWNjb3VudCIzIm5hdHMiOnsibGltXRz
    Ijp7InN1YnMiOi0xLCJjb25uIjotMSwibGVhZiI6LTESImltcG9ydHMiOi0xLCJleHBvcnRzIjotMSwiZGF0YSI6LTESIn
    BheWxvYWQiOi0xLCJ3aWxkY2FyZHMhOnRydWV9fX0._WW5C1triCh8a4jhyBxEZZP8RJ17pINS8qLzz-
    01o6zbz1uZfT0JGvwST56Yv2_849B9iUXSd-8kp1iMXHdoBA
}
```

The `MEMORY` resolver is recommended when the server has a small number of accounts that don't change very often.

For more information on how to configure a memory resolver, see [this tutorial](#).

URL Resolver

NOTE: The [standalone NATS Account JWT Server](#) is now *legacy*, please use the [NATS Based Resolver](#) instead. However, the URL resolver option is still available in case you want to implement your own version of an account resolver

The `URL` resolver specifies a URL where the server can append an account public key to retrieve that account's JWT. Convention for standalone NATS Account JWT Servers is to serve JWTs at: `http://localhost:9090/jwt/v1/accounts/`. For such a configuration, you would specify the resolver as follows:

```
resolver: URL(http://localhost:9090/jwt/v1/accounts/)
```

“ Note that if you are not using a nats-account-server, the URL can be anything as long as by appending the public key for an account, the requested JWT is returned.

If the server used requires client authentication, or you want to specify which CA is trusted for the lookup of account information, specify `resolver_tls`. This [tls configuration map](#) lets you further restrict TLS to the resolver.

#3

□ cocobook-sync □ 2026-10-28 11:14:09 UTC

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