

Basics



- [Creating an Operator, Account and User](#)
 - [NSC Assets](#)
 - [The NSC Stores Directory](#)
 - [The NKEYS Directory](#)
 - [Listing Keys](#)
 - [The Operator JWT](#)
 - [The Account JWT](#)
 - [The User JWT](#)
 - [Account Server Configuration](#)
 - [NATS Server Configuration](#)
 - [Pushing the local nsc changes to the nats server](#)
 - [Client Testing](#)
 - [Create a nats context](#)
 - [NSC Embeds NATS tooling](#)
 - [User Authorization](#)
 - [The NSC Environment](#)
-

NSC allows you to manage identities. Identities take the form of *nkeys*. Nkeys are a public-key signature system based on Ed25519 for the NATS ecosystem.

The nkey identities are associated with NATS configuration in the form of a JSON Web Token (JWT). The JWT is digitally signed by the private key of an issuer forming a chain of trust. The `nsc` tool creates and manages these identities and allows you to deploy them to a JWT account server, which in turn makes the configurations available to nats-servers.

There's a logical hierarchy to the entities:

- `Operators` are responsible for running nats-servers, and issuing account JWTs. Operators set the limits on what an account can do, such as the number of connections, data limits, etc.
- `Accounts` are responsible for issuing user JWTs. An account defines streams and services that can be exported to other accounts. Likewise, they import streams and services from

other accounts.

- `Users` are issued by an account, and encode limits regarding usage and authorization over the account's subject space.

NSC allows you to create, edit, and delete these entities, and will be central to all account-based configuration.

In this guide, you'll run end-to-end on some of the configuration scenarios:

- Generate NKey identities and their associated JWTs
- Make JWTs accessible to a nats-server
- Configure a nats-server to use JWTs

Let's run through the process of creating some identities and JWTs and work through the process.

Creating an Operator, Account and User

Let's create an operator called `MyOperator`.

There is an additional switch `--sys` that sets up the system account which is required for interacting with the NATS server. You can create and set the system account later.

```
nsc add operator MyOperator
```

```
[ OK ] generated and stored operator key
"ODSWWTKZLRDFBPXNMNAY7XB2BIJ45SV756BHUT7GX6JQH6W7AHVAFX6C"
[ OK ] added operator "MyOperator"
[ OK ] When running your own nats-server, make sure they run at least version 2.2.0
```

With the above command, the tool generated an NKEY for the operator, stored the private key safely in its keystore.

Lets add a service URL to the operator. Service URLs specify where the nats-server is listening. Tooling such as `nsc` can make use of that configuration:

```
nsc edit operator --service-url nats://localhost:4222
```

```
[ OK ] added service url "nats://localhost:4222"
[ OK ] edited operator "MyOperator"
```

Creating an account is just as easy:

```
nsc add account MyAccount
```

```
[ OK ] generated and stored account key  
"AD2M34WBNGQFYK37IDX53DPRG74RLLT7FFWB0BMBUXMAVBCVAU5VKWIY"  
[ OK ] added account "MyAccount"
```

As expected, the tool generated an NKEY representing the account and stored the private key safely in the keystore.

Finally, let's create a user:

```
nsc add user MyUser
```

```
[ OK ] generated and stored user key  
"UAWBXL SZVZHNDIURY52F6WETFCFZLXYUEFJAHRXDW7D2K4445IY4BVXP"  
[ OK ] generated user creds file `~/.nkeys/creds/MyOperator/MyAccount/MyUser.creds`  
[ OK ] added user "MyUser" to account "MyAccount"
```

As expected, the tool generated an NKEY representing the user, and stored the private key safely in the keystore. In addition, the tool generated a *credentials* file. A credentials file contains the JWT for the user and the private key for the user. Credential files are used by NATS clients to identify themselves to the system. The client will extract and present the JWT to the nats-server and use the private key to verify its identity.

NSC Assets

NSC manages three different directories:

- The nsc home directory which stores nsc related data. By default nsc home lives in `~/.nsc` and can be changed via the `$NSC_HOME` environment variable.
- An *nkeys* directory, which stores all the private keys. This directory by default lives in `~/.nkeys` and can be changed via the `$NKEYS_PATH` environment variable. The contents of the nkeys directory should be treated as secrets.
- A *stores* directory, which contains JWTs representing the various entities. This directory lives in `$NSC_HOME/nats`, and can be changed using the command `nsc env -s <dir>`. The stores directory can be stored under revision control. The JWTs themselves do not contain any secrets.

The NSC Stores Directory

The stores directory contains a number of directories. Each named by an operator in question, which in turn contains all accounts and users:

```
tree ~/.nsc/nats
```

```
/Users/myusername/.nsc/nats
├── MyOperator
│   ├── MyOperator.jwt
│   └── accounts
│       ├── MyAccount
│       │   ├── MyAccount.jwt
│       │   └── users
│       │       └── MyUser.jwt
```

These JWTs are the same artifacts that the NATS servers will use to check the validity of an account, its limits, and the JWTs that are presented by clients when they connect to the nats-server.

The NKEYS Directory

The nkeys directory contains all the private keys and credential files. As mentioned before, care must be taken to keep these files secure.

The structure keys directory is machine friendly. All keys are sharded by their kind **O** for operators, **A** for accounts, **U** for users. These prefixes are also part of the public key. The second and third letters in the public key are used to create directories where other like-named keys are stored.

```
tree ~/.nkeys
```

```
/Users/myusername/.nkeys
├── creds
│   ├── MyOperator
│   │   ├── MyAccount
│   │   └── MyUser.creds
├── keys
│   ├── A
│   │   └── DE
│   │       └── ADETPT36WBIBUKM3IBCV4A5YUSDXFEJPW4M6GGVBYCBW7RRNFTV5NGE.nk
│   ├── O
│   │   └── AF
│   │       └── 0AFEYYSZYVI4FXLRJTMM32PQEI3RGOWZJT7Y3YFM4HB7ACPE4RTJPG.nk
└── U
```

└─ DB

└─ UDBD5FNQPSLI06CDMIS5D4EBNFKYWVDNULQTFTUZJXWFNYLGFF52VZN7.nk

The `nk` files themselves are named after the complete public key, and stored in a single string - the private key in question:

```
cat ~/.nkeys/keys/U/DB/UDBD5FNQPSLI06CDMIS5D4EBNFKYWVDNULQTFTUZJXWFNYLGFF52VZN7.nk
```

```
SUAG35IAY2EF5D0ZRV6MUS0FDGJ602BQCZHSRPLIK6J3GVCX366BFAYSNA
```

The private keys are encoded into a string, and always begin with an `S` for *seed*. The second letter starts with the type of key in question. `O` for operators, `A` for accounts, `U` for users.

In addition to containing keys, the `nkeys` directory contains a `creds` directory. This directory is organized in a way friendly to humans. It stores user credential files or `creds` files for short. A credentials file contains a copy of the user JWT and the private key for the user. These files are used by NATS clients to connect to a NATS server:

```
cat ~/.nkeys/creds/MyOperator/MyAccount/MyUser.creds
```

```
-----BEGIN NATS USER JWT-----
eyJ0eXAiOiJKV1QiLCJhbGciOiJI1ZDI1NTE5LW5rZXkifQ.eyJqdGkiOiI0NUc3MkhIQUVCRFBQV05ZWktMTUuHjYWFYFRSSUVDQ1NVQUI2VDZRUjdVM1JZUFZaM05BIiwiaWF0IjoxNjM1Mzc1NTYxLCJpc3MiOiJBRDJNMzRXQk5HUUZZSzM3SURYN
TNEUFJHNzRSTExUN0ZGV0JPQk1CVVhNQVZCQ1ZBVTVWS1dJWSIsIm5hbWUiOiJNeVZzZXIiLCJzdWIiOiJlVQVdCWExTWlZ
aSE5ESVVSwtUyRjZXRVRGQ0ZaTFhZVUUGSkFIUlhEVzdEMks0NDQ1SVk0Q1ZYUCIsIm5hdHMiOiJp7fSwic3ViI
jp7fSwic3VicyI6LTEsImRhdGEiOi0xLCJwYXlsb2FkIjotMSwidHlwZSI6InVzZXIiLCJ2ZXJzaW9uIjoyfX0.CGymhGY
HfdZyhUeucxNs9TthSjy_27LVZikqsvm-pLili8KNe1xy0Vnk_w-xPWdrCx_t3Se2lgXmoy3wBcVCw
-----END NATS USER JWT-----
```

***** IMPORTANT *****

NKEY Seed printed below can be used to sign and prove identity.

NKEYs are sensitive and should be treated as secrets.

```
-----BEGIN USER NKEY SEED-----
```

```
SUAP2AY6UAWHOXJBWDNRNKJ2DHNC5VA2DFJZTF6C6PMLKUCOS2H2E2BA2E
```

```
-----END USER NKEY SEED-----
```

Listing Keys

You can list the current entities you are working with by doing:

```
nsc list keys
```

```
+-----+
-+
|
Keys                                     |
+-----+-----+-----+-----+
+-----+
| Entity      | Key                                     | Signing Key | Stored
|
+-----+-----+-----+-----+
+-----+
| MyOperator  | ODSWTKZLRDFBPXNMNAY7XB2BIJ45SV756BHUT7GX6JQH6W7AHVAFX6C |           |
*           |
| MyAccount   | AD2M34WBNGQFYK37IDX53DPRG74RLLT7FFWB0BMBUXMAVBCVAU5VKWIY |           |
*           |
| MyUser      | UAWBXL SZVZHNDIURY52F6WETFCFZLXYUEFJAHRXDW7D2K4445IY4BVXP |           |
*           |
+-----+-----+-----+-----+
+-----+
```

The different entity names are listed along with their public key, and whether the key is stored. Stored keys are those that are found in the nkeys directory.

In some cases you may want to view the private keys:

```
nsc list keys --show-seeds
```

```
+-----+
|                                     Seeds Keys                                     |
+-----+-----+-----+-----+
| Entity      | Private Key                                     | Signing Key |
+-----+-----+-----+-----+
| MyOperator  | SOAJ3JDZBE6JKJ0277CQP5RIAA7I7HBI44RDCMTIV3TQRYQX350TXSMHAE |           |
| MyAccount   | SAAACXWSQIKJ4L2SEAUZJR3BCNSRCN32V5UJSABCSEP35Q7LQRPV6F4JPI |           |
| MyUser      | SUAP2AY6UAWH0XJBWDNRNKJ2DHNC5VA2DFJZTF6C6PMLKUC0S2H2E2BA2E |           |
+-----+-----+-----+-----+
[ ! ] seed is not stored
[ERR] error reading seed
```

If you don't have the seed (perhaps you don't control the operator), nsc will decorate the row with a `!`. If you have more than one account, you can show them all by specifying the `--all` flag.

The Operator JWT

You can view a human readable version of the JWT by using `nsc`:

```
nsc describe operator
```

```
+-----+
|                                     |
|                               Operator Details                               |
|                                     |
+-----+-----+
| Name                | MyOperator                |
| Operator ID         | ODSWWTkZLRDFBPxNMNAY7XB2BIJ45SV756BHUT7GX6JQH6W7AHVAFX6C |
| Issuer ID          | ODSWWTkZLRDFBPxNMNAY7XB2BIJ45SV756BHUT7GX6JQH6W7AHVAFX6C |
| Issued             | 2021-10-27 22:58:28 UTC   |
| Expires            |                           |
| Operator Service URLs | nats://localhost:4222     |
| Require Signing Keys | false                     |
+-----+-----+
```

Since the operator JWT is just a JWT you can use other tools, such as jwt.io to decode a JWT and inspect its contents. All JWTs have a header, payload, and signature:

```
{
  "typ": "jwt",
  "alg": "ed25519"
}
{
  "jti": "ZP2X3T2R57SLXD2U5J30LLYIVW2LFBMTXRPMMGISQ50F7LANUQPQ",
  "iat": 1575468772,
  "iss": "0AFEEYZSYVVI4FXLRXJTMM32PQEI3RGOWZJT7Y3YFM4HB7ACPE4RTJPG",
  "name": "0",
  "sub": "0AFEEYZSYVVI4FXLRXJTMM32PQEI3RGOWZJT7Y3YFM4HB7ACPE4RTJPG",
  "type": "operator",
  "nats": {
    "operator_service_urls": [
      "nats://localhost:4222"
    ]
  }
}
```

```
}  
}
```

All NATS JWTs will use the `algorithm` `ed25519` for signature. The payload will list different things. On our basically empty operator, we will only have standard JWT `claim` fields:

`jti` - a jwt id `iat` - the timestamp when the JWT was issued in UNIX time `iss` - the issuer of the JWT, in this case the operator's public key `sub` - the subject or identity represented by the JWT, in this case the same operator `type` - since this is an operator JWT, `operator` is the type

NATS specific is the `nats` object, which is where we add NATS specific JWT configuration to the JWT claim.

Because the issuer and subject are one and the same, this JWT is self-signed.

The Account JWT

Again we can inspect the account:

```
nsc describe account
```

```
+-----+  
|                                     Account Details                                     |  
+-----+  
| Name                               | MyAccount                               |  
| Account ID                         | AD2M34WBNGQFYK37IDX53DPRG74RLLT7FFWB0BMBUXMAVBCVAU5VKWIY |  
| Issuer ID                         | ODSWWTkZLRDFBPXNMNAY7XB2BIJ45SV756BHUT7GX6JQH6W7AHVAFX6C |  
| Issued                           | 2021-10-27 22:59:01 UTC               |  
| Expires                           |                                         |  
+-----+  
| Max Connections                    | Unlimited                               |  
| Max Leaf Node Connections          | Unlimited                               |  
| Max Data                           | Unlimited                               |  
| Max Exports                        | Unlimited                               |  
| Max Imports                        | Unlimited                               |  
| Max Msg Payload                    | Unlimited                               |  
| Max Subscriptions                  | Unlimited                               |  
| Exports Allows Wildcards           | True                                    |  
| Response Permissions               | Not Set                                |  
+-----+  
| Jetstream                          | Disabled                                |
```

+-----+-----+		
Imports	None	
Exports	None	
+-----+-----+		

The User JWT

Finally the user JWT:

```
nsc describe user MyUser
```

+-----+-----+		
	User	
+-----+-----+		
Name	MyUser	
User ID	UAWBXL SZVZHNDIURY52F6WETFCFZLXYUEFJAHRXDW7D2K4445IY4BVXP	
Issuer ID	AD2M34WBNGQFYK37IDX53DPRG74RLLT7FFWB0BMBUXMAVBCVAU5VKWIY	
Issued	2021-10-27 22:59:21 UTC	
Expires		
Bearer Token	No	
Response Permissions	Not Set	
+-----+-----+		
Max Msg Payload	Unlimited	
Max Data	Unlimited	
Max Subs	Unlimited	
Network Src	Any	
Time	Any	
+-----+-----+		

The user id is the public key for the user, the issuer is the account. This user can publish and subscribe to anything, as no limits are set.

When a user connects to a nats-server, it presents it's user JWT and signs a nonce using its private key. The server verifies if the user is who they say they are by validating that the nonce was signed using the private key associated with the public key, representing the identify of the user. Next, the server fetches the issuer account and validates that the account was issued by a trusted operator completing the chain of trust verification.

Let's put all of this together, and create a simple server configuration that accepts sessions from U

Account Server Configuration

To configure a server to use accounts, you need to configure it to select the type of *account resolver* it will use. The preferred option being to configure the server to use the built-in [NATS Based Resolver](#).

NATS Server Configuration

If you don't have a nats-server installed, let's do that now:

```
go get github.com/nats-io/nats-server
```

Let's create a configuration that references our operator JWT and the nats-account-server as a resolver. You can use `nsc` itself to generate the security part of the server configuration that you can just add to your `nats-server` config file.

For example to use the NATS resolver (which is the recommended resolver configuration) use `nsc generate config --nats-resolver`.

Edit this generated configuration as needed (e.g. adjust the location where the server will store the JWTs in `resolver.dir`) and paste it into your nats-server configuration (or save it to a file and import that file from within you server config file).

At minimum, the server requires the `operator` JWT, which we have pointed at directly, and a resolver.

e.g.

```
nsc generate config --nats-resolver > resolver.conf
```

And example server config `myconfig.cfg`

```
server_name: servertest
listen: 127.0.0.1:4222
http: 8222

jetstream: enabled

include resolver.conf
```

Now start this local test server using `nats-server -c myconfig.cfg`

The nats-server requires a designated account for operations and monitoring of the server, cluster, or supercluster. If you see this error message:

```
nats-server: using nats based account resolver - the system account needs to be specified in configuration or the operator jwt
```

Then there is no system account to interact with the server and you need to add one to the configuration or operator JWT. Let's add one to the operator JWT using `nsc`:

```
nsc add account -n SYS`  
nsc edit operator --system-account SYS
```

(and re-generate `resolver.conf`)

Now start the local test server using: `nats-server -c myconfig.cfg`

Pushing the local nsc changes to the nats server

In order for the nats servers to know about the account(s) you have created or changes to the attributes for those accounts, you need to push any new accounts or any changes to account attributes you may have done locally using `nsc` into the built-in account resolver of the nats-server. You can do this using `nsc push`:

For example to push the account named 'MyAccount' that you have just created into the nats server running locally on your machine use:

```
nsc push -a MyAccount -u nats://localhost
```

You can also use `nsc pull -u nats://localhost` to pull the view of the accounts that the local NATS server has into your local nsc copy (i.e. in `~/nsc`)

As soon as you 'push' an the account JWT to the server (that server's built-in NATS account resolver will take care of distributing that new (or new version of) the account JWT to the other nats servers in the cluster) then the changes will take effect and for example any users you may have created with that account will then be able to connect to any of the nats server in the cluster using the user's JWT.

Client Testing

Install the `nats` CLI Tool if you haven't already.

Create a subscriber:

```
nats sub --creds ~/.nkeys/creds/MyOperator/MyAccount/MyUser.creds ">"
```

Publish a message:

```
nats pub --creds ~/.nkeys/creds/MyOperator/MyAccount/MyUser.creds hello NATS
```

Subscriber shows:

```
Received on [hello]: 'NATS'
```

Create a `nats` context

If you are going to use those credentials with `nats` you should create a context so you don't have to pass the connection and authentication arguments each time:

```
nats context add myuser --creds ~/.nkeys/creds/MyOperator/MyAccount/MyUser.creds
```

NSC Embeds NATS tooling

To make it easier to work, you can use the NATS clients built right into NSC. These tools know how to find the credential files in the keyring. For convenience, the tools are aliased to `sub`, `pub`, `req`, `reply`:

```
nsc sub --user MyUser ">"
...

nsc pub --user MyUser hello NATS
...
```

See `nsc tool -h` for more detailed information.

User Authorization

User authorization, as expected, also works with JWT authentication. With `nsc` you can specify authorization for specific subjects to which the user can or cannot publish or subscribe. By default a user doesn't have any limits on the subjects that it can publish or subscribe to. Any message stream or message published in the account is subscribable by the user. The user can also publish to any subject or imported service. Note that authorization, if configured, must be specified on a per user basis.

When specifying limits it is important to remember that clients by default use generated "inboxes" to allow publish requests. When specifying subscribe and publish permissions, you need to enable clients to subscribe and publish to `_INBOX.>`. You can further restrict it, but you'll be responsible for segmenting the subject space so as to not break request-reply communications between clients.

Let's say you have a service that your account clients can make requests to under `q`. To enable the service to receive and respond to requests it requires permissions to subscribe to `q` and publish permissions under `_INBOX.>`:

```
nsc add user s --allow-pub "_INBOX.>" --allow-sub q
```

```
[ OK ] added pub pub "_INBOX.>"
[ OK ] added sub "q"
[ OK ] generated and stored user key
"UDYQFIF75SQU2NU3TG4JXJ7C5LFCWAPXX5SSRB276YQ00FXHFIGHXMEL"
[ OK ] generated user creds file `~/nkeys/creds/MyOperator/MyAccount/s.creds`
[ OK ] added user "s" to account "MyAccount"
```

```
nsc describe user s
```

+-----+-----+-----+		
	User	
+-----+-----+-----+		
Name	s	
User ID	UDYQFIF75SQU2NU3TG4JXJ7C5LFCWAPXX5SSRB276YQ00FXHFIGHXMEL	
Issuer ID	AD2M34WBNGQFYK37IDX53DPRG74RLLT7FFWB0BMBUXMAVBCVAU5VKWIY	
Issued	2021-10-27 23:23:16 UTC	
Expires		
Bearer Token	No	
+-----+-----+-----+		
Pub Allow	_INBOX.>	

Sub Allow	q	
Response Permissions	Not Set	
+-----+-----+		
Max Msg Payload	Unlimited	
Max Data	Unlimited	
Max Subs	Unlimited	
Network Src	Any	
Time	Any	
+-----+-----+		

As you can see, this client is now limited to publishing responses to `_INBOX.>` addresses and subscribing to the service's request subject.

Similarly, we can limit a client:

```
nsc add user c --allow-pub q --allow-sub "_INBOX.>"
```

```
[ OK ] added pub pub "q"
[ OK ] added sub "_INBOX.>"
[ OK ] generated and stored user key
"UDIRTIVVHCW2FLLDHTS27ENXLVNP4E04Z5MR7FZUNXFXWREPGQJ4BRRE"
[ OK ] generated user creds file `~/nkeys/creds/MyOperator/MyAccount/c.creds`
[ OK ] added user "c" to account "MyAccount"
```

Lets look at that new user

```
nsc describe user c
```

+-----+-----+		
	User	
+-----+-----+		
Name	c	
User ID	UDIRTIVVHCW2FLLDHTS27ENXLVNP4E04Z5MR7FZUNXFXWREPGQJ4BRRE	
Issuer ID	AD2M34WBNGQFYK37IDX53DPRG74RLLT7FFWB0BMBUXMAVBCVAU5VKWIY	
Issued	2021-10-27 23:26:09 UTC	
Expires		
Bearer Token	No	
+-----+-----+		
Pub Allow	q	
Sub Allow	_INBOX.>	
Response Permissions	Not Set	

+-----+-----+-----+		
Max Msg Payload	Unlimited	
Max Data	Unlimited	
Max Subs	Unlimited	
Network Src	Any	
Time	Any	
+-----+-----+-----+		

The client has the opposite permissions of the service. It can publish on the request subject `q`, and receive replies on an inbox.

The NSC Environment

As your projects become more involved, you may work with one or more accounts. NSC tracks your current operator and account. If you are not in a directory containing an operator, account or user, it will use the last operator/account context.

To view your current environment:

```
nsc env
```

```
+-----+
-----+
|                                     NSC
Environment                               |
+-----+-----+
+-----+
| Setting          | Set | Effective
Value                                     |
+-----+-----+
+-----+
| $NSC_CWD_ONLY    | No  | If set, default operator/account from cwd
only               |
| $NSC_NO_GIT_IGNORE | No  | If set, no .gitignore files
written            |
| $NKEYS_PATH      | No  |
~/nkeys            |
| $NSC_HOME        | No  |
~/nsc              |
```

```

| Config | |
~/.nsc/nsc.json |
| $NATS_CA | No | If set, root CAs in the referenced file will be used for nats
connections |
| | | If not set, will default to the system trust
store |
+-----+-----+
+-----+
| From CWD | |
No |
| Stores Dir | |
~/.nsc/nats |
| Default Operator | |
MyOperator |
| Default Account | |
MyAccount |
| Root CAs to trust | | Default: System Trust
Store |
+-----+-----+
+-----+

```

If you have multiple accounts, you can use `nsc env --account <account name>` to set the account as the current default. If you have defined `NKEYS_PATH` or `NSC_HOME` in the environment, you'll also see their current effective values. Finally, if you want to set the stores directory to anything other than the default, you can do `nsc env --store <dir containing an operator>`. If you have multiple accounts, you can try having multiple terminals, each in a directory for a different account.

```

[] #1
[] cocobook-sync [] 2026-10-28 11:10:48 UTC
[] cocobook-sync [] 2026-10-28 11:10:49 UTC

```