Securing Your NATS Deployment -- Hands-on Lab Guide

RethinkConn 2026 workshop. This is the attendee follow-along. Every step is a command you run yourself, followed by the output you should see, so you can confirm each step worked before moving on. If your output matches, you are good; if it does not, see Troubleshooting.

You will build a NATS cluster's security from the ground up: static credentials, decentralized JWT with `nsc`, and auth callout (where you change a user's permissions live without restarting the server), then package the identity store for import into Synadia Control Plane.

Before you start (prerequisites)

You need the following tools on your machine. Everything else (`nsc`, `nats-server`, key generation) runs inside Docker, so you do not install it.

Tool	What for	Install
Docker + Compose v2	runs the NATS servers and tooling	docker.com/get-started
`nats` CLI	publishing test messages	see below
`make`	lab targets	any version; macOS: `xcode-select --install`; Linux: `apt install make`
`python3`	Lab 3 flip-bob step	ships with macOS/most Linux; python.org
`curl`	Makefile health poll	ships with macOS/most Linux

Windows: the labs need bash and make, so they run inside WSL2 -- which you already have if Docker Desktop is installed (WSL2 is its default backend). Native PowerShell/cmd is not supported. One-time setup:

1. Docker Desktop -> Settings -> Resources -> WSL integration -> enable it for your distro (Ubuntu by default). No distro yet? `wsl --install` in an admin PowerShell, then reboot.

2. Open the distro (run `wsl` in any terminal) and install the host tools:

```shell
sudo apt-get update && sudo apt-get install -y make python3 curl git
curl -sf https://binaries.nats.dev/nats-io/natscli/nats@latest | sh
sudo mv nats /usr/local/bin/
```

3. Clone the repo inside your WSL home directory (for example `~/`), not under `/mnt/c` -- run all lab commands from that WSL shell.

Install the `nats` CLI (macOS / Linux):

```shell
# macOS
brew install nats-io/nats-tools/nats
# Linux / generic (the installer downloads to the CURRENT directory)
curl -sf https://binaries.nats.dev/nats-io/natscli/nats@latest | sh
sudo mv nats /usr/local/bin/
```

Self-check -- run these before the first lab:

```shell
docker compose version      # expect: v2 or later (any modern Docker Desktop)
nats --version              # expect: 0.4.0 (or later)
make --version              # expect: GNU Make 3.x or later
python3 --version           # expect: 3.x
```

Pre-pull all images before the session (takes ~350 MB) so you are not downloading during the workshop:

```shell
make prefetch               # pre-pulls all images (~350 MB) -- run this before
the session
```

If all self-checks pass, you are ready. Then grab the bundle and change into it:

```shell
git clone https://github.com/synadia-labs/nats-security-workshop
cd nats-security-workshop
make help                          # lists every target
```

> **Ports:** each scenario publishes on its own host port (00 = 4222/8222, 01 = 4322/8322, 02 = 4422/8422), so you can run more than one at a time without a collision.

Lab 1 -- Static credentials (Block 3a)

The simplest model: usernames and passwords (or NKeys) in the server config. `alice` has full permissions; `bob` may only use `test.>`.

```shell
make 00
```

Expected output (the last lines):

```
 Container workshop-baseline Started
--- user/password mode ---
alice  pub app.foo      OK
alice  pub test.foo     OK
bob    pub app.foo      DENIED
bob    pub test.foo     OK

Expect: alice succeeds on both; bob denied on app.foo, OK on test.foo
```

`bob` is denied on `app.foo` because his permissions are scoped to `test.>`. Now swap passwords for cryptographic NKeys (same permission model, no shared secret on the wire):

```shell
make 00-nkey
```

Expected output:

```
None
gen-nkeys: alice NKey = U...
gen-nkeys: bob   NKey = U...
gen-nkeys: seeds written to creds/alice.seed and creds/bob.seed
 Container workshop-baseline Recreate
 Container workshop-baseline Recreated
 Container workshop-baseline Starting
 Container workshop-baseline Started
--- NKey mode ---
alice  pub app.foo      OK
alice  pub test.foo     OK
bob    pub app.foo      DENIED
bob    pub test.foo     OK

Expect: alice succeeds on both; bob denied on app.foo, OK on test.foo
```

The point: identical authorization, but every change is a config reload and a re-distribution of secrets. Tear down before the next lab: `make 00-down`.

Lab 2 -- Decentralized JWT with nsc (Block 3b)

Now identity lives in signed JWTs, not the server config. The server trusts only the operator key; accounts and users are distributed through the NATS resolver.

```
Shell
make 01
```

Expected output (the tail):

```
None
bootstrap: creating operator WorkshopOp + system account...
bootstrap: adding APP account...
bootstrap: adding users alice (full) and bob (test.> only)...
bootstrap: extracting operator.jwt for server mount...
bootstrap: extracting account JWTs for resolver preload...
bootstrap: copying user creds for easy nats CLI access...
bootstrap: done.
  operator JWT:  server-state/operator.jwt
  resolver JWTs: server-state/jwt/{SYS,APP}.jwt
  user creds:    creds/{alice,bob}.creds
--- JWT mode (creds files) ---
```

```
alice  pub app.foo      OK
alice  pub test.foo     OK
bob    pub app.foo      DENIED
bob    pub test.foo     OK

Expect: alice succeeds on both; bob denied on app.foo, OK on test.foo
```

Same allow/deny result, but `bob`'s limits now live inside his signed user JWT. Decode it and read the claims:

Shell

```
make 01-decode            # alice (default)
WHO=bob make 01-decode    # bob
```

`alice` has no publish/subscribe restriction (full account access). `bob`'s token carries the scope:

None

```
+----------------------+------------------------------------------------------------
+
| Name                 | bob
|
| User ID              | U... (generated fresh each run -- yours will differ)
|
+----------------------+------------------------------------------------------------
+
| Pub Allow            | test.>
|
| Sub Allow            | test.>
|
+----------------------+------------------------------------------------------------
+
```

The point: permissions, limits, and expiry travel inside the signed token. Rotating `bob` is a JWT operation, not a server restart. Tear down: `make 01-down`.

Lab 3 -- Auth callout (Block 3c) -- the flagship, on your keys

Here identity comes from outside NATS. The server forwards each connection request to a callout service that decides yes/no and signs the user JWT. You will change `bob`'s permissions live, with **no NATS server restart**.

```Shell
make 02
```

This builds the callout service and brings up the stack. Expected tail:

```None
 Container workshop-callout Started
Waiting for callout service to register with NATS...
--- alice=full, bob=[test.>] (current callout/users.json) ---
alice  pub app.foo      OK
alice  pub test.foo     OK
bob    pub app.foo      DENIED
bob    pub test.foo     OK
```

Now widen `bob` to `app.>` and restart only the callout container:

```Shell
make 02-flip-bob
```

Expected output:

```None
flip-bob: users.json updated:
16c16,17
<       "test.>"
---
>       "test.>",
>       "app.>"
19c20,21
<       "test.>"
---
>       "test.>",
>       "app.>"
```

```
flip-bob: restarting ONLY the callout container (NATS server stays up)...
 Container workshop-callout Restarting
 Container workshop-callout Started

--- Round 2: bob's perms widened to app.> ---
--- alice=full, bob=[test.>, app.>] (current callout/users.json) ---
alice  pub app.foo      OK
alice  pub test.foo     OK
bob    pub app.foo      OK
bob    pub test.foo     OK

Inspect the callout's decisions with: make 02-logs
```

The unified diff shows `app.>` being added to both `pub` and `sub` arrays for bob (two hunks: lines 16 and 19 in users.json).

`bob` now publishes to `app.foo`. **The NATS server was never restarted, no operator key was touched, no creds were redistributed** -- the callout service made the new decision on `bob`'s next connection. Watch the decisions stream by in another pane with `make 02-logs`. Tear down: `make 02-down`.

> **Reset note:** re-running `make 02` after flip-bob shows bob already allowed on `app.>` because `users.json` was mutated. To reset: run `cp 02-auth-callout/callout/users.baseline.json 02-auth-callout/callout/users.json` or run `make clean` (which does this automatically).

Lab 4 -- Package for Synadia Control Plane (Block 4)

> **Prerequisite:** `make 03` packages the nsc store created by Lab 2. Run `make 01` (or `make 01-bootstrap` if you do not want the server running) before this lab.

You built an operator/account/user store by hand. Package it so it can be imported into Synadia Control Plane as a Bring-Your-Own system.

```
Shell
make 03
```

Expected output (abridged):

```
None
package-nsc: wrote .../03-scp-import/nsc-store.tar.gz
package-nsc: contents:
  nsc-store/nats/nsc/stores/WorkshopOp/WorkshopOp.jwt
  nsc-store/nats/nsc/stores/WorkshopOp/accounts/APP/APP.jwt
  nsc-store/nats/nsc/stores/WorkshopOp/accounts/APP/users/alice.jwt
  nsc-store/nats/nsc/stores/WorkshopOp/accounts/APP/users/bob.jwt
  nsc-store/nkeys/creds/WorkshopOp/APP/alice.creds
  ...
```

Your listing will be ~45 lines; the block above shows only the key files. As long as you see `WorkshopOp.jwt`, `APP.jwt`, `alice.jwt`, `bob.jwt`, `alice.creds`, `bob.creds`, you are good.

> **Heads-up:** the tarball includes the private account and operator key seeds (`nsc-store/nkeys/keys/...nk`). That is what the Control Plane import needs to take ownership of the accounts, and it is why these are throwaway demo keys. Never share a real `nsc` store tarball outside a trusted boundary.

This packaging step is the runnable part of Lab 4. Importing the tarball into Synadia Control Plane (and walking the resulting management experience) is demonstrated during the session.

Inspect the nsc store for SCP import (make 03-decode)

```
Shell
make 03-decode
```

This prints the operator JWT, SYS account JWT, and key seeds from the Lab 2 nsc store formatted for pasting into Synadia Control Plane's BYO (Bring-Your-Own) import flow.

Expected output (JWT and seed strings abbreviated with `...`; everything after the seed-type prefix varies per run):

```
None
+-------------------------------------------------------------------------------
------+
|                                 Operator Details
|
+----------------------+--------------------------------------------------------
------+
```

```
| Name                  | WorkshopOp
|
| Operator ID           | O... (generated fresh each run -- yours will differ)
|
...
+-----------------------+-------------------------------------------------------
------+

+-------------------------------------------------------------------------------
-----+
|                                  Account Details
|
+---------------------------+-------------------------------------------------
-----+
| Name                        | SYS
|
...
+---------------------------+-------------------------------------------------
-----+

JWTs for Import


Operator

eyJ0eXAiOiJKV1Qi...

SYS Account

eyJ0eXAiOiJKV1Qi...

Seed

+-------------------------------------------------------------------------------
------+
|                                    Seeds Keys
|
+------------+-----------------------------------------------------------+-------
------+
| Entity     | Private Key                                               |
Signing Key |
+------------+-----------------------------------------------------------+-------
------+
| WorkshopOp | SOA...                                                    |
|
|  SYS       | SAA...                                                    |
|
```

```
|  SYS       | SAA...                                                        | *
|
|   sys      | SUA...                                                        |
|
+------------+---------------------------------------------------------------+-------
------+
[ ! ] seed is not stored
[ERR] error reading seed
[ T ] auth callout encryption target
```

The `[ ! ] seed is not stored` and `[ERR] error reading seed` lines are **expected** -- the callout encryption key (`[ T ]`) is not held in this nsc store and nsc correctly reports it missing. This is not a failure.

> **Warning:** the output contains live private key seeds. That is fine for this throwaway workshop operator. **Never share output from a real nsc store.**

After importing: Control Plane pushes updated account JWTs to your NATS server over the network, and it cannot reach a server on your laptop's localhost. So changes you make in the Control Plane UI (new users, rotations) will not appear on your local workshop server unless you expose it (for example via ngrok). Creating accounts, adding users, and rotating keys inside Control Plane all work -- that is the point of this exercise. On a real deployment your servers have reachable URLs and this push is automatic.

Clean up

```Shell
make clean        # tears down all stacks and removes generated creds
```

Expected: every `workshop-*` container removed, ending with `Cleaned.`

Troubleshooting

Symptom	Fix
`make` step hangs on "NATS did not become ready"	Docker is slow to pull the first time; re-run the same target. Confirm Docker is running with `docker ps`.
Port already in use	Another scenario is up on that port. Tear it down (`make 00-down` / `01-down` / `02-down`) or relocate. Lab 1 (most likely to collide -- 4222 is the default NATS port): `make 00 NATS_PORT_00=5222 MON_PORT_00=9222`. Lab 2: `make 01 NATS_PORT_01=5322 MON_PORT_01=9322`.
`nats: command not found`	The `nats` CLI is not on your PATH; re-run the install one-liner above.
`make 02` fails to build	Needs network for the first image build; re-run. The Docker build is the only step that reaches the internet.
You fell behind	Each lab is independent. Skip ahead with the matching `make 0N`; you do not need the previous lab still running. (Exception: Lab 4 packages Lab 2's store -- run `make 01` first.)

Build/run reference for presenters lives in README.md. This file is the attendee-facing follow-along.