# NATS Configuration Contexts

| Metadata | Value |
|---|---|
| Date | 2021-12-14 |
| Author | @ripienaar |
| Status | Partially Implemented |
| Tags | client |

## Background

A `nats context` is a named configuration stored in a configuration file allowing a set of related configuration items to be stored and accessed later.

In the `nats` CLI this is used extensively, for example `nats stream ls --context orders` would load the `orders` context and configure items such as login credentials, servers, domains, API prefixes and more.

The intention of the ADR is to document the storage of these contexts so that clients can, optionally, support using them.

## Version History

| Date | Revision |
|---|---|
| 2020-08-12 | Initial basic design |
| 2020-05-07 | JetStream Domains |
| 2021-12-13 | Custom Inbox Prefix |
| 2024-12-03 | Windows Cert Store, User JWT and TLS First |

This reflects a current implementation in use widely via the CLI as such it's a stable release. Only non breaking additions will be considered.

## Design

Today the design is entirely file based for maximum portability, later we can consider other options like S3 buckets, KV stores etc.

### Configuration Paths

There is generally no standard for what goes on in a users home directory on a Unix system. Recently the Free Desktop team have been working on XDG Base Directory Specification that specifies in detail where configuration, data, binaries and more are to be stored in a way that's compatible with Linux desktops like KDE and Gnome but also with systems such as systemd.

We therefore based the design on this specification as a widely supported standard.

| File | Description |
|---|---|
| `~/.config` | The default location for user configuration, configurable using `XDG_CONFIG_HOME` |
| `~/.config/nats` | Where all NATS related user configuration should go |
| `~/.config/nats/context.txt` | The current selected (default) context, would contain just `ngs` |
| `~/.config/nats/context/ngs.json` | The configuration for the `ngs` context |

While this is Linux centered it does work on Windows, we might want to consider a more typical path to replace `~/.config` there and keep the rest as above.

### Context content

The `~/.config/nats/context/ngs.json` file has the following JSON fields:

| Key | Default | Description |
|---|---|---|
| `description` | | A human friendly description for the specific context |
| `url` | `nats://localhost:4222` | Comma seperated list of server urls |
| `token` | | Authentication token |
| `user` | | The username to connect with, requires a password |
| `password` | | Password to connect with |
| `creds` | | Path to a NATS Credentials file |
| `nkey` | | Path to a NATS Nkey file |
| `cert` | | Path to the x509 public certificate |
| `key` | | Path to the x509 private key |
| `ca` | | Path to the x509 Certificate Authority |
| `nsc` | | A `nsc` resolve url for loading credentials and server urls |
| `jetstream_domain` | | The JetStream Domain to use |
| `jetstream_api_prefix` | | The JetStream API Prefix to use |
| `jetstream_event_prefix` | | The JetStream Event Prefix |
| `inbox_prefix` | | A prefix to use when generating inboxes |
| `user_jwt` | | The user JWT token |
| `tls_first` | | Enables the use of TLS on Connect rather than historical INFO first approach |

| Key | Default | Description |
|---|---|---|
| `windows_cert_store` | | The Windows cert store to use for access to the TLS files, `windowscurrentuser` or `windowslocalmachine` |
| `windows_cert_match_by` | | Which certificate to use inside the store |
| `windows_cert_match` | | How certificates are searched for in the store, `subject` or `issuer` |
| `windows_ca_certs_match` | | Which Certificate Authority to use inside the store |

All fields are optional, none are marked as `omitempty`, users wishing to edit these with an editor should known all valid key names.

Above settings map quite obviously to client features with the exception of `nsc`, the `nsc` key takes a URL like value, examples are:

- `nsc://operator`
- `nsc://operator/account`
- `nsc://operator/account/user`
- `nsc://operator/account/user?operatorSeed&accountSeed&userSeed`
- `nsc://operator/account/user?operatorKey&accountKey&userKey`
- `nsc://operator?key&seed`
- `nsc://operator/account?key&seed`
- `nsc://operator/account/user?key&seed`
- `nsc://operator/account/user?store=/a/.nsc/nats&keystore=/foo/.nkeys`

The context invokes `nsc generate profile <url>`, the responce will be non zero exit code for error else a structure like:

```
{
  "user_creds": "<filepath>",
  "operator" : {
     "service": "hostport"
   }
}
```

This will configure the `creds` and `url` parts of the context. If either `url` or `creds` is specifically configured in the Context those will override the answer from `nsc`.

See `nsc generate profile --help` for details.

## Sample Usage APIs

I don't think we really want to dictate standard APIs here, in Go we have 2 main ways to use the package.

It's also fine to delegate management of these to the `nats` CLI.

A very basic and quick way to just connect to a specific context:

```
nc, _ := Connect(os.GetEnv("CONTEXT"), nats.MaxReconnects(-1))
```

Here Connect will construct `[]nats.Option` based on the context settings and append the user supplied ones to the list.

A way to load a context and optionally override some options:

```
nctx, _ := Load(os.GetEnv("CONTEXT"), WithServerURL("nats://other:4222"))
nc, _ := nctx.Connect(nats.MaxReconnects(-1))
```

When `name` is empty it will use the current selected context, if no context is selected and name is empty it's an error.