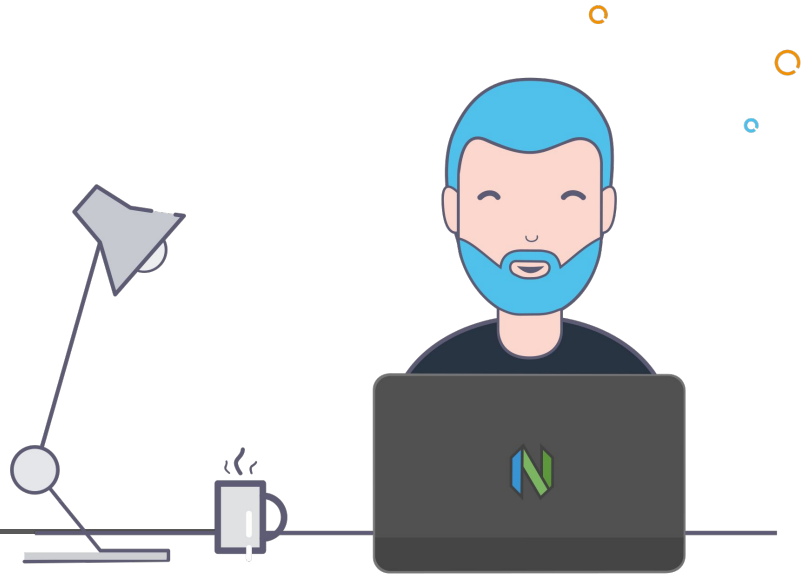
Config, config everywhere

How to compose the configuration and secrets of microservices taking into account various variables without dying in the attempt

Who I am?

- *Algorithm and DS enthusiast*
- *Backend engineer since 2017*
- *FOSDEM since 2018*
- *Mainly DB related contributions*



Blog: https://gonlo2.github.io/blog/

Github: @Gonlo2

1. Dynamic configuration

Dynamic configuration definition

Dynamic configuration is the ability to change the behavior and functionality of a running system without requiring application restarts. An ideal dynamic configuration system enables service developers and administrators to view and update configurations easily, and delivers configuration updates to the applications efficiently and reliably.

Dynamic configuration at Twitter
https://blog.twitter.com/engineering/en_us/topics/infrastructure/2018/dynamic-configuration-at-twitter.html

Dynamic configuration MHConfig

MHConfig is an experiment that you could define as an efficient way to compose and obtain configuration easily, and it main features are:

- Atomic
- Label-based
- Allow references and overrides
- Secure
- Traceable
- Efficient

https://github.com/Gonlo2/mhconfig

Dynamic configuration MHConfig

For example you could ask for the configuration "*databases*" with the labels `{app: sessions, env: prd, datacenter: belgium, secrets: prd}` and you defined the configurations:

- Doc "database", labels `{}` →
  `{db: "{params/db/prefix}{params/db/db_name}", user: "{params/db/prefix}{params/db/user}"}`
- Doc "params", labels `{app: sessions}` → `{db: {db_name: sessions, user: sessions}}`
- Doc "params", labels `{env: prod}` → `{db: {prefix: prod_}}`
- Doc "database", labels `{secrets: prd, app: sessions, env: prod, datacenter: belgium}` →
  `{passwd: XXXXXXX}`
- Doc "database", labels `{env: prod, datacenter: belgium}` →
  `{writer_hosts: [10.1.1.128], reader_hosts: [10.1.1.128]}`

The returned config will be:

```
db: prod_sessions
user: prod_sessions
passwd: XXXXXXX
writer_hosts: [10.1.1.128]
reader_hosts: [10.1.1.128]
```

2. Overview

Configuration directory structure

The configuration is defined in a directory from which you can request documents from any subpath and whose path and file names have a special meaning for the daemon, following the next rules:

- Any file/directory starting with a dot is ignored.
- The path specifies the document labels and it can have two formats:
  - Compact if the folder follows the scheme `key=value` where `key` is the key of the label and `value` its value.
  - Otherwise if follow the scheme `key/value` where a first level specifies the key and the next level its value.
- A file starting with a underscore is a special file.
  - The file type is the string from the beginning to the first dot, the current ones are `_text` for text files and `_bin` for binary files.
- A file with the extension `.yaml` is recognized as a configuration file where the document name is the string from the beginning to the last dot.
  - For example the file `database.yaml` is the document `database`.
- Other files are ignored.

Configuration labels

The labels allow define specific characteristics of the configuration to be composed, for example the type of environment, the name of the service, the type of secrets, etc.

The algorithm used to calculate the order of the configurations is as follows.

- All stored configurations that are a subset of the requested labels are obtained.
- Those configurations are first ordered by the number of labels that form it, from less to more specialization.
- For those configurations with the same number of labels, an ordered list with the weights of the label keys is created and the configurations are ordered with respect to this list.

For this it is necessary to have a configuration with metadata of the labels as their weight, this content has to be stored in the `mhconfig.yaml` document of the root of the configuration.

Configuration tags

The YAML configuration allow some custom tags to facilitate the development.

- `!format` allow do a string interpolation with references to the config.
  - The path of the config value to obtain is defined between brackets, using the slash to separate the key values.
  - To escape `{` and `}` use `{{` and `}}`.
  - `!format "https://{domains/api/web}/api/v1/config"` obtain the value in `api/web` of the document `domains` and format the string with it.
- `!ref` insert the configuration of another file if don't exists a circular dependency.
  - `!ref [database, schema]` to reference the value in `schema` of the document `database`.
- `!sref` insert a scalar or a null value from the same configuration file.
  - `!sref [message]` to reference the value in `message` of the self document.
- `!delete` remove the previous element with that path.
- `!override` force override one value instead merge it.
- `!!str` , `!!binary` , `!!int` , `!!float` , `!!bool` defines the type of a scalar.

ACL tokens

A token allows to authenticate an entity and configure some authentication parameters. The tokens are defined in the `tokens.yaml` root file of the configuration folder as a YAML file, this file has the following structure:

```
tokens:
- value: root
  expire_at: 0
  labels:
    kind: root
- value: dev
  expire_at: 0
  labels:
    kind: developer
    env: dev
- value: calendar-prod
  expire_at: 0
  labels:
    kind: app
    env: prod
    app: calendar
```

ACL policies

A policy allows restrict access to resources according to the capability, the root path or the labels.

```
capabilities: [GET, WATCH, TRACE]
root_paths:
- path: /mnt/data/mhconfig/+
  capabilities: [GET, WATCH, TRACE]
labels:
- key: test
  value: me
  capabilities: [GET, WATCH]
- key: test
  capabilities: [GET, WATCH]
- capabilities: [GET, WATCH, TRACE]
```

Where the character `+` denote any value within a single path segment and the character `*` denote any number of path segments (can only be used at the end of the path).

API

A gRPC API is used to communicate with the daemon and it allows:

- Obtain a specific configuration version of a document.
- Register to get the last configuration version of a document.
- Notify configuration update on a path.
- Trace the actions of the server according to some filters.

To use it you could use the next existing clients:

- A python script to notify the changes in a configuration path using inotify.
- Using grpcurl.
- Writing and sharing your own client ;)

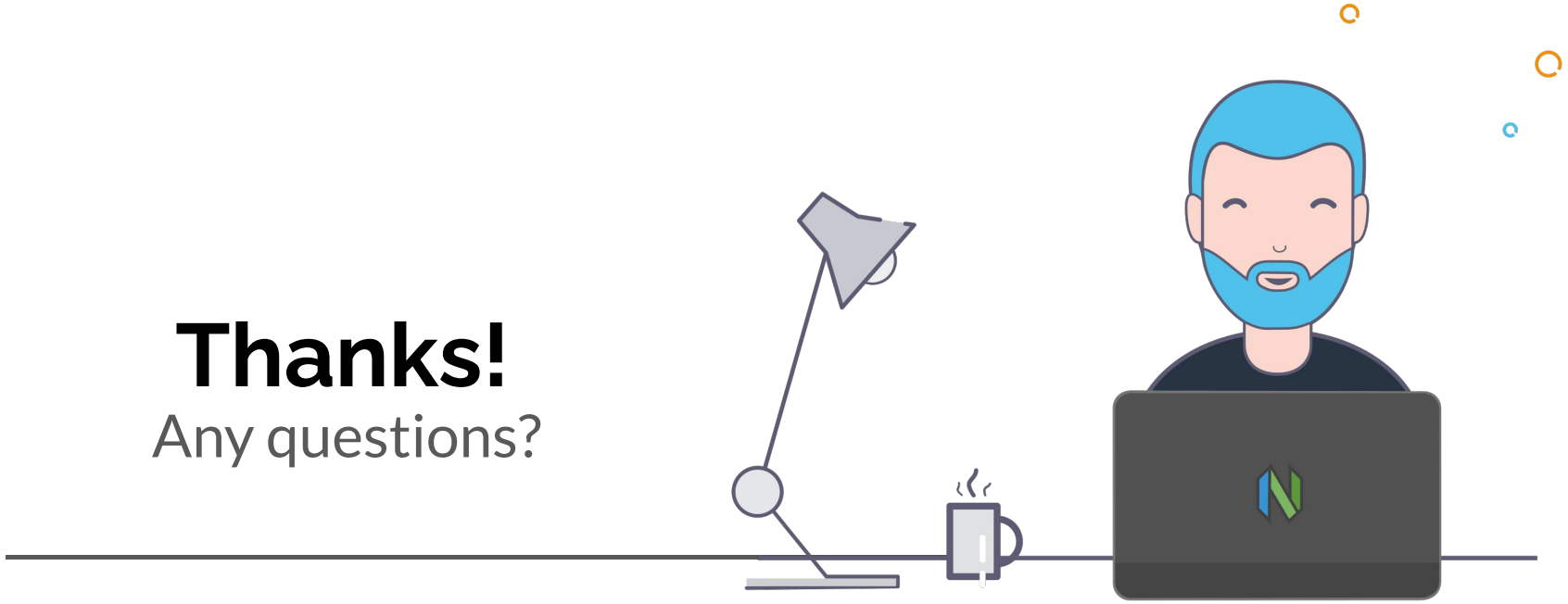
3. Demo

Things to improve

- Create a configuration for the daemon where store number of threads, grpc credentials, etc.
- Create some clients
- Create some git ops updater
- Investigate his use like a standalone tool, as a k8s config object, etc.
- Improve the provided information by the trace request
- Allow ask for multiple documents in one request
- Deduplicate the returned config with the watch command
- Fix some localized bugs and do cleaning tasks

Thanks!

Any questions?



Blog: https://gonlo2.github.io/blog/

Github: @Gonlo2