PLCS Report - Starship Toolset

Table of Contents

Introduction

Overview

Nebula is "a scalable overlay networking tool" which allows you to "seamlessly connect computers anywhere in the world". [1] It uses UDP hole punching to allow nodes to connect directly even if they are behind a firewall which only allows established traffic through.

To run Nebula on a node, you must have the `nebula` binary, along with a private key, certificate and yaml config file. Signing certificates and customising the config files can become tedious when the size of a Nebula network grows beyond a few nodes. Often users will create keypairs and certificates for all nodes from a single host, then transfer the private keys and certificates to the correct nodes. This goes against best security practices as it involves transferring a private key, often across a network, and it means that a host, other than the node which will use the key, has had access to this private key.

This toolkit aims to overcome some of these issues by making it easy to bring up a new node, provisioning a certificate and giving it configuration, without the private key leaving the node.

Although Nebula can scale to support thousands of nodes, this toolkit is currently focused on (but not limited to) managing smaller networks such as homelab networks - where some hosts are based on a home network, some may be running in 'the cloud' with VPS services, and some nodes such as laptops and mobiles may be constantly moving between different private networks.

The 'Starship' toolkit includes an API server (Quasar) with a database which acts as a central management system, a client tool (Neutron) used by nodes to request to join networks and update their certificates and configuration, and a web client (Hubble) which communicates with the API server in order to manage networks. The management system can support multiple networks, and the client tool will allow a node to join multiple networks. The management system will sign certificates for nodes when they have been approved using the API.

A demo of this project can be found here: https://youtu.be/gIlgz1huZPI

Alternatives

One alternative to designing a new system for certificate signing would be to create an extension or a fork of the `step-ca` certificate management tool. This would be very powerful and useful for large datacentres as it would integrate with many different forms of authentication and identity services. It is likely that there would be large groups of nodes which could use the same configuration, meaning tools such as Ansible, Chef or Puppet would be able to set up and give the configuration files to the correct nodes.

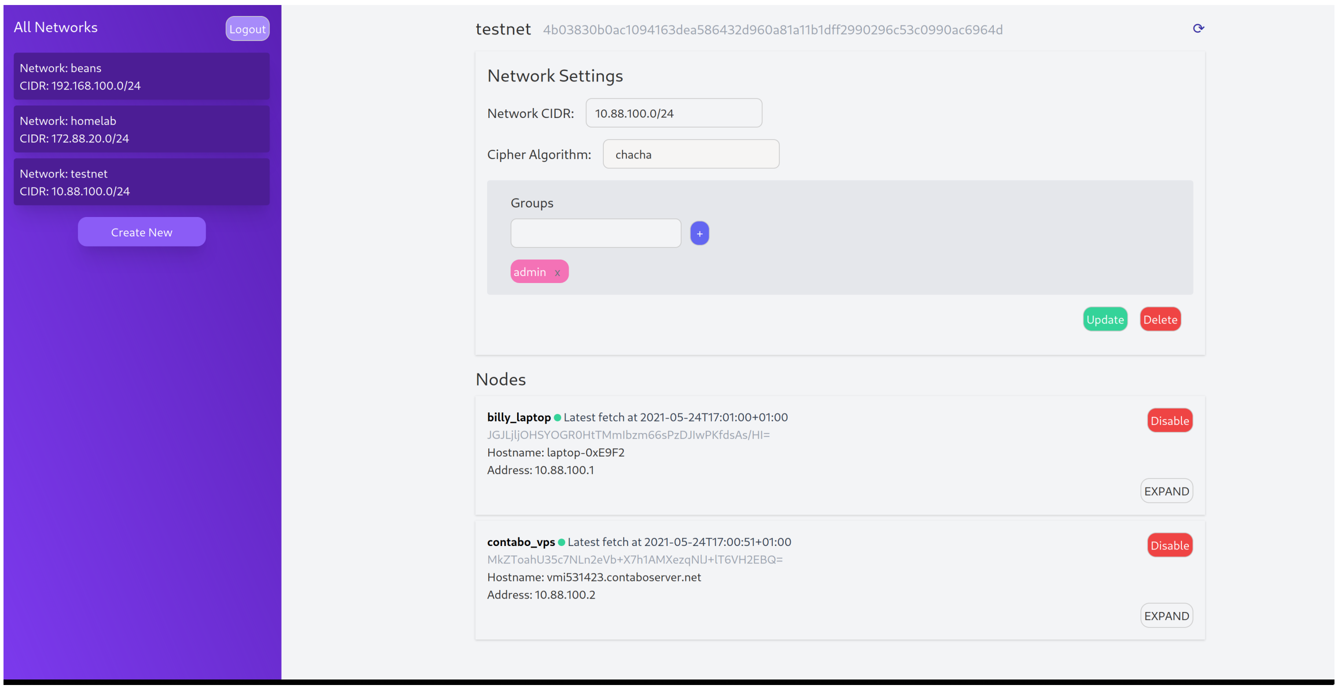
However this would lose the ability to have fine grained control over configuration. For smaller networks, this control can be very useful where each node has a specific purpose and therefore need different firewall rules.

Additionally, `step-ca` is now a large and mature project which would take time to understand well enough to successfully add the ability to sign Nebula certificates. Considering that I'm working with a language that I have never used before, it made more sense to write a new program from scratch, albeit less complex and powerful.

Hubble

Overview

Hubble is a frontend application which communicates with the Quasar API in order to manage Starship networks. It shows all available networks in a sidebar, in addition to a 'Create New' button. When you select a network it shows network settings, which you can modify. You also have the option to delete a network.


All Networks
Logout

Network: beans
CIDR: 192.168.100.0/24

Network: homelab
CIDR: 172.88.20.0/24

Network: testnet
CIDR: 10.88.100.0/24

Create New

testnet 4b03830b0ac1094163dea586432d960a81a11b1dff2990296c53c0990ac6964d

Network Settings

Network CIDR: 10.88.100.0/24

Cipher Algorithm: chacha

Groups

+

admin x

Update Delete

Nodes

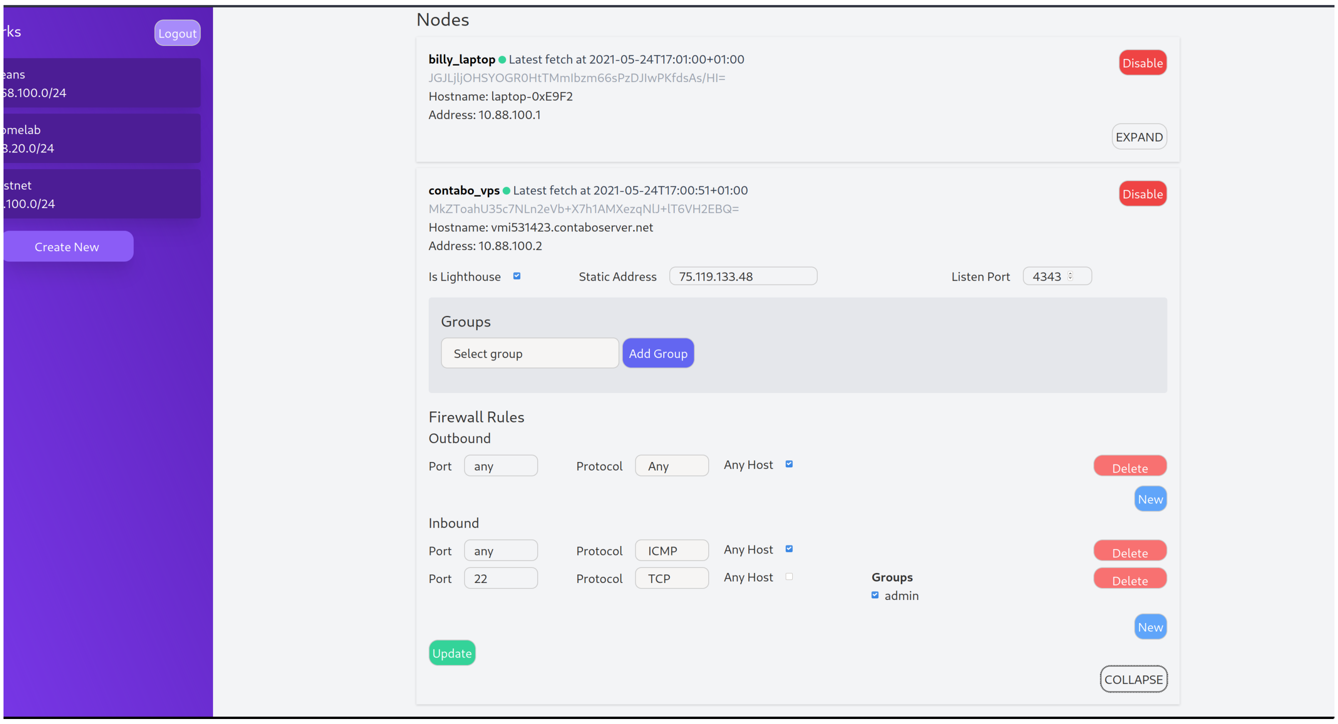
billy_laptop ● Latest fetch at 2021-05-24T17:01:00+01:00
JGJLjljOHSYOGR0HtTMmIbzm66sPzDJIwPKfdsAs/HI=
Hostname: laptop-0xE9F2
Address: 10.88.100.1
Disable
EXPAND

contabo_vps ● Latest fetch at 2021-05-24T17:00:51+01:00
MkZToahU35c7NLn2eVb+X7h1AMXezqNlJ+lT6VH2EBQ=
Hostname: vmi531423.contaboserver.net
Address: 10.88.100.2
Disable
EXPAND

Figure 1. Hubble Network Page

The network settings page also shows all nodes in the network as collapsible cards, which initially show simple information such as the nodename, hostname and IP address, but can be expanded to reveal settings for the node that can be updated such as firewall rules, and groups which the node is in etc.

Hubble Node Management


ks
Logout

eans
68.100.0/24

omelab
3.20.0/24

stnet
.100.0/24

Create New

Nodes

billy_laptop ● Latest fetch at 2021-05-24T17:01:00+01:00
JGJLjljOHSYOGR0HtTMmIbzm66sPzDJIwPKfdsAs/HI=
Hostname: laptop-0xE9F2
Address: 10.88.100.1
Disable
EXPAND

contabo_vps ● Latest fetch at 2021-05-24T17:00:51+01:00
MkZToahU35c7NLn2eVb+X7h1AMXezqNlJ+lT6VH2EBQ=
Hostname: vmi531423.contaboserver.net
Address: 10.88.100.2
Disable

Is Lighthouse ☑ Static Address 75.119.133.48 Listen Port 4343

Groups
Select group | Add Group

Firewall Rules
Outbound
Port any Protocol Any Any Host ☑ Delete
New

Inbound
Port any Protocol ICMP Any Host ☑ Delete
Port 22 Protocol TCP Any Host ☐ Groups ☑ admin Delete
New

Update
COLLAPSE

Language and Paradigm Chosen

To make the application easily accessible for users, I have built it as a web application. This means it works cross platform and does not require any installation on a client device. Hubble is built with HTML, CSS and JavaScript as this is what is needed to make an interactive web application.

In order to develop more efficiently, Hubble is built using a tool called 'Svelte'. Although this can be considered an alternative to frameworks such as React, Angular and Vue, it is not a framework as such but more of a compiler. It allows you to program reactively, with code broken down into components. The coding process is similar to when using frameworks such as React, but code is compiled to static HTML, CSS and native JavaScript - whereas many other frameworks bundle a full library which the client uses to interpret the code at runtime. This allows Svelte to have a small footprint in terms of both resource usage and size of the compiled site.

Svelte uses a reactive programming paradigm, which is a subset of declarative paradigms. You can declare what should happen as a result of something else - to make the frontend **react** to changes as they happen. For example if a change is made to a network's settings, other parts of the interface can **react** and update to reflect the change.

Installation Instructions

```
cd hubble
npm install

# to build to static site (not needed for running dev server)
npm run build
```

Operating Instructions

The build creates a `public` directory containing HTML, CSS and JavaScript files which can be served using any HTTP server.

```
cd hubble

# to run dev server
npm run dev

# to run 'production' server
npm run start
```

Libraries and Tools Needed to Run

- nodejs
- npm - node package manager
    - svelte - compiler for `.svelte` files to HTML/CSS/JS
    - svelte-notifications - for notifications

- axios - for API requests to Quasar
- svelte-routing - For managing pages and navigation
- tailwindcss - simple class based css framework

Issues

There were no major issues with the development of the Hubble frontend, but it involved the challenge of learning a new style of web app development as I had never used Svelte before.

Coming from a React background meant I had to learn new concepts of global stores, event management and overall project management. However, Svelte is simple and easy to learn so I was able to pick these up reasonably quickly.

Neutron

Overview

Neutron is a client which Starship nodes use to request to join networks, and update their configuration and certificates.

When joining a new network, Neutron will create a new Nebula keypair. It will then send a request to Quasar to join a specific network. This request includes the node name, the network it wants to join, its hostname and its Nebula public key. This information is sent as a JSON payload, signed using the Nebula private key. This is encoded similarly to a PASETO token. PASETO tokens are similar to JSON Web Tokens (JWTs), however do not suffer the same vulnerabilities JWTs suffer due to the vague protocol specification.

When updating, Neutron will send requests to Quasar to obtain an updated certificate and configuration file. For Quasar to send these, Neutron must include a signed token which includes it’s nodename and the network name it is trying to update, and the node must be approved and active on the Quasar server. The signature on the token is verified against the public key stored for the node on the Quasar server.

The update script can be run at frequent intervals to keep the node updated with the most recent configuration changes.

Language and Paradigm Chosen

Neutron is written in Golang. There were many reasons for this, but the most significant is that Golang can statically compile binaries easily. This means that a small binary can be downloaded to a node with no extra dependencies required to use the tool.

Golang has many other advantages. For example, it is strongly typed, and there is little 'magic' as with languages such as Python. The go compiler is also 'fussy'. For example, it will refuse to compile when you have an unused variable declared. Although this makes it harder to work with initially, it means it is easier to write good code.

Golang is an imperative language, but it supports programming in object oriented and functional paradigms. An imperative language is necessary due to the complexity and unique nature of the tools. Features of object oriented programming such as classes and inheritence are not available in golang, but other features including polymorphism (using interfaces) and methods are available and have been used in this tool.

Installation Instructions

```
# build
cd starship

# equivalent of `go build -o neutron cmd/neutron/*.go`
make neutron
```

Operating Instructions

Manual install

```
# request to join network
./neutron join -quasar http://127.0.0.1:6947 -network NETWORK -name
NAME

# approve node from frontend then fetch latest config from Quasar
./neutron update -network NETWORK
# send SIGHUP to nebula to force config reload
pgrep nebula | xargs sudo kill -1
```

Using Install Script

```
# quick install from release
wget https://github.com/b177y/starship-
public/releases/download/v0.3.0/install-neutron.sh -O /tmp/install-
neutron.sh

# check content
less /tmp/install-neutron.sh
bash /tmp/install-neutron.sh

# approve node from frontend then fetch latest config from Quasar
neutron update -network NETWORK

# start nebula with systemd
sudo systemctl start nebula@NETWORK

# send SIGHUP to nebula to force config reload
pgrep nebula | xargs sudo kill -1
```

Libraries and Tools Needed to Run

- Golang
  - slackhq/nebula - nebula certificate tools
  - sirupsen/logrus - logging library
  - tetris-io/shortid - library for creating short uuids
- systemd (not a hard requirement, but used for example setup)
- Nebula - this is provided by the install script but otherwise must be downloaded from here

Issues

The keys used by nebula are saved in the Montgomery format as they are used for x25519 Diffie-Helman key exchange. This means they cannot be used to sign standard PASETO tokens - which can only use ed25519 signatures for asymmetric key authentication. This requires Edwards formatted keys rather than Montgomery. The "twisted Edwards curve used by Ed25519 and the Montgomery curve used by X25519 are birationally equivalent" [2] which means you can convert between the two key formats. However you can only convert directly from Edwards to Montgomery, not the other way around.

To avoid having multiple private keys for each network a node is in (one for Nebula and one for communicating with Quasar), I created a library for signing and verifying 'XPASETO' tokens. These use Montgomery keys for XEdDSA signatures, outlined by Signal. [3] This package is based off an existing paseto library, [4] from which functions are borrowed where it wasn't necessary to rewrite them. It should be noted that the XPASETO library does NOT conform with the PASETO standard (see https://paseto.io/rfc/ section 5.2).

Quasar

Overview

Quasar is a Central Management System (CMS) for managing Starship networks. It provides APIs for two types of clients:

- Neutron Nodes
  - These authenticate by signing requests using their nebula private key
- Frontend clients / management tools
  - These authenticate using JSON Web Tokens

Quasar can be configured using a yaml config file. By default the API listens on port `6947` as the Helix Nebula is 694.7 light years away from earth.

The API for neutron nodes provides the following endpoints:

- /api/neutron/join - for a node to request to join a network. This request includes the Nebula public key for the node. The request is signed by the corresponding private key. This self-signed request is verified by Quasar.
- /api/neutron/update - for a node to request configuration information and a certificate. Quasar will work out the configuration options based off the node's config in the database, and the

config of other nodes.

The API for management clients provides endpoints for:

- listing networks
- getting CA cert for a network
- listing nodes in a network
- updating network settings
- updating node settings
- approving / enabling / disabling nodes

Language and Paradigm Chosen

Quasar is written in Golang, for many of the same reasons as Neutron. In addition to these reasons, Nebula tools and libraries are written in Golang. Nebula has a custom certificate format (not x509 or SSH certs) and slack have made the library for interacting with these certificates open source so it is easy to include in a project.

Although it would be possible to use the `nebula-cert` commandline tool with other languages using subprocesses, this would be less clean and less efficient than importing and using the native functions needed. Using a language other than Go would have to add this as an extra dependency for the tool.

Furthermore, I could not find a way to use Montgomery keys for XEdDSA signatures in Python (the most likely alternative to Golang for this tool), and writing the cryptography functions from scratch myself would be a security (and time management) risk as maths and cryptography are not my areas of expertise. Golang has well maintained cryptography libraries as part of the language's standard package. Using the built in libraries in addition to some code borrowed from third party libraries, I was able to write a JSON token signing library which uses XEdDSA signatures.

Golang uses an imperative programming paradigm. See the Neutron section for more on this.

Installation Instructions

```
make quasar
```

Operating Instructions

```
# set JWT signing secret
export QUASAR_AUTHSECRET=$(uuid)

# set admin account password
export QUASAR_ADMINPASS="password"

# start server
./quasar serve -config examples/quasar.yml
```

Libraries and Tools Needed to Run

- Golang
    - slackhq/nebula - nebula certificate tools
    - boltdb/bolt - embedded key/value database
    - gorilla/mux - http router
    - urfave/negroni - http middleware manager
    - meatballhat/negroni-logrus - logging middleware support
    - sirupsen/logrus - logging library
    - rs/cors - CORS middleware

Issues

Part way through the project I decided to rewrite Quasar in Python, as I am more familiar with Python and I was running into time constraints. I had rewritten most of the API in Python when I tried to replicate the XPASETO library I had earlier written in Golang. I was unable to find the necessary libraries in Python to support this. Although Golang is a newer language than Python, it was created by Google and has always had a focus on security, meaning the built in crypto libraries are more advanced.

Another problem I had with the Python rewrite was that I had to use the `nebula-cert` binary with subprocesses for creating and signing certificates. This adds an extra dependency to the project and is not a clean way of interacting with certificates.

I decided to switch back to Golang for these reasons, but fortunately I ended up finding it easier than I thought it would be.

Another problem I had was with the conversion of Edwards keys (used by the CA) to Montgomery Curve25519 keys (used by Nebula nodes). I used functions from a project by Filippo Valsorda (Go team security lead) to perform the key conversion. [5] The function for converting public keys worked, but the private key function did not. After lots of research, I found that key clamping [6] was needed.

Conclusions

The state of the toolset as of the submission is fully functional and fulfils all requirements from the project plan.

The toolset can be used to create and manage Nebula overlay networks. A demo of this (see introduction) shows that the tools successfully work together.

Future Work

Improved Authentication

Currently there is a single user called 'admin', and the password is defined by the environment variable `QUASAR_ADMINPASS`. This is good enough for basic demonstrations and usage for a simple network such as a homelab which is managed by one person. Others can still join nodes to the network, while one person can manage access.

However, for larger scale networks such as those of small corporations or academic groups, it would be useful for multiple people to be able to manage networks. This could involve role based access e.g. one user can manage all nodes in a specific network while another user can only manage a specific node.

Additionally, there is no authentication on the Quasar endpoint that Neutron uses to request to join a network. This is not a direct security risk to the network as nodes must be approved by an authorised client before they can receive a certificate signed by the CA. However, if Quasar is running on the Internet, denial of service attacks could be possible as someone could repeatedly request to join a network.

A possible solution is for a token to be created for each network which would be required with a join request. These tokens could be rotated at intervals such as every 24 hours. The tokens would be less sensitive than credentials for the management endpoints as nodes would still require approval. This means you could share tokens with people who you partially trust so that they can join your network, and you wouldn't have to worry about them changing firewall rules in your network. You would then only be risking a denial of service from these people you partially trust, which is a significantly smaller attack surface compared to being open to the internet.

HTTPS Support

Currently HTTPS can be set up using a reverse proxy such as traefik or nginx. Using tools such as docker, this can be set up quickly and easily with a replicable setup. However, one of the big advantages of using Golang is that it compiles code to a static binary. Golang's built in HTTP server (which Quasar uses) has support for running over HTTPS. This means that it would be very easy to add support.

To use HTTPS you would run the server with:

```
http.ListenAndServeTLS(addr, certFile, keyFile, handler)
```

Instead of:

```
http.ListenAndServe(addr, handler)
```

Adding built in support would involve adding an option to the yaml config to enable HTTPS, and to configure key and cert paths.

Input Validation

When incoming JSON is decoded by the Quasar API, it isn't validated against any constraints. This means injection attacks could be attempted, for example it is possible to create a network called "<script>alert(0)</script>".

Although Svelte protects against this and renders the script tag as a string, it should be validated by Quasar to limit the possibilities of injection attacks.

Golang structs allow 'tags' on attributes, which are used by the json package to decode and encode json data. Third party libraries provide the ability to use additional tags to add validators to these tags.

For example when creating a new network, the `NewNetSchema` struct is used.

```go
type NewNetSchema struct {
    Name string `json:"name"`
    Cidr string `json:"cidr"`
}
```

The `json:"name"` tag tells the json decoder if there is a field with the key `name`, it should use the value as the value for `NewNetSchema.Name`. Using an external library such as `go-playground/validator` you could add validators as follows:

```go
type NewNetSchema struct {
    Name string `json:"name" validate:"max=30,alphanum"`
    Cidr string `json:"cidr" validate:"cidrv4"`
}
```

You could then validate requests using:

```go
// example test struct
net := NewNetSchema{
    Name: "testnet",
    Cidr: "192.168.1.0/24",
}
err := validate.Struct(net)
if err := nil {
    log.Error(err)
}
```

Overall, although the current project fulfils its requirements and works as intended, there are lots of improvements that can be made to improve the security and usability of the toolset.

Appendices

Changelog

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

0.3.0 (2021-05-23)

Added

- Quasar signs config using XPASETO tokens (additional security layer to HTTPS)

- Network certificate fingerprints are shown in Hubble frontend
- Node public key fingerprints are shown in Hubble frontend
- JWT Middleware has been added to require username+password auth to manage networks.

0.2.0 (2021-05-22)

Added

- Added ability to modify firewall rules through Quasar
- Added firewall update forms to Hubble
- Time is saved when a node fetches the latest config
- Latest config fetch time is shown in Hubble

0.1.0 (2021-05-22)

2021-05-22

Added

- Listen port can now be changed through frontend

Fixed

- Formatting of static host map in nebula yaml config

2021-05-21

Added

- Quasar config endpoint fully working
- Neutron get config and write to yaml file working

2021-05-20

Added

- Update node endpoint working in Quasar
- Update node functionality working in hubble

2021-05-19

Changed

- Using negroni golang library for logging and future authentication middleware.

Added

- Added cipher type to network information in db and API responses.
- Completed update network API endpoint.

- Added node info endpoints.
- Added groups array to networks in db and made them updatable.

2021-05-18

Added

- Started working on Hubble frontend using Svelte compiler
- Created frontend app structure and integrated with quasar
- Modified how CORS requests work with quasar to work with client

2021-05-17

Added

- Finished neutron join network capability
- Started adding neutron update capability

2021-05-16

Added

- Node Endpoints for Quasar
- Finished key network endpoints for Quasar
- Added some neutron endpoints

2021-05-14

Changed

- Removed Python and switched back to golang due to crypto dependencies

2021-05-10

Changed

- Using Python for Quasar instead of golang

Initial Stages - 2021-04-08

Added

- Project Structure
    - `cmd` directory for neutron and quasar main outputs
    - `examples` for example config
    - `nebutils` as library for neutron and quasar to share code
    - general repo files e.g. README, LICENSE, CHANGELOG
- Started work on `quasar`

    - Added functionality to add a new network
    - Added basic http server functionality
    - added ability to interact with boltdb as a database interface
- Started work on `neutron`
    - Added main function to parse user flags
    - Started on `init.go` which gets ca cert, generates keys and requests signing

References

[1] GitHub. 2021. slackhq/nebula. [online] Available at: https://github.com/slackhq/nebula

[2] Valsorda, F. 2019. Using Ed25519 signing keys for encryption [online] Available at: https://blog.filippo.io/using-ed25519-keys-for-encryption/

[3] Perrin, T. 2016. The XEdDSA and VXEdDSA Signature Schemes [online] Available at: https://signal.org/docs/specifications/xeddsa/

[4] GitHub. 2021. o1egl/paseto. [online] Available at: https://github.com/o1egl/paseto

[5] GitHub. 2021. FiloSottile/age. [online] Available at: https://github.com/FiloSottile/age/blob/bbab440e198a4d67ba78591176c7853e62d29e04/internal/age/ssh.go#L174

[6] Craige, J. 2021. An Explainer On Ed25519 Clamping [online] Available at: https://www.jcraige.com/an-explainer-on-ed25519-clamping