KUB 31: Плагины в Helm на примере плагина шифрования данных

Описание:

Итак, сейчас нам предстоит разобраться с то и дело встречающейся проблемой в ежедневной жизни - хранением секретов в системе контроля версий. Подумайте, что вы сделали helm чарт, который устанавливает ваше приложение в кубер кластер. Ваше приложение скорее всего будет использовать какие-то environment переменные, чтобы подключиться к базе или каким-то другим сервисам. Очевидно, что записывать логин и пароль в открытом виде в values.yaml не станет хорошей идеей - ведь в итоге вы зальете свой чарт в систему контроля версий и пароль смогут увидеть все желающие. Поэтому нам необходимо как-то научиться шифровать эти данные. Для этого мы разделим наш рассказ на две части. В первой обсудим как в принципе мы можем шифровать любые файлы, а во второй посмотрим на то как мы можем интегрировать это решение с helm.
SOPS
Итак, начнем мы с sops. По сути, sops - это редактор, который позволяет вам шифровать различные типы файлов и расшифровывать их, используя различные приватные ключи или сервисы для шифрования. Давайте установим sops и посмотрим на пример.
SOPS написан на golang, поэтому вы можете использовать go и собрать пакет на любой системе. Но мы пойдем более простым путем и скачаем уже скомпилированный бинарник. Список для скачивания можно найти на странице релизов:

```
# wget
https://github.com/mozilla/sops/releases/download/v3.7.1/sops-v3
.7.1.linux
# mv sops-v3.7.1.linux /usr/local/bin/sops
# chmod +x /usr/local/bin/sops
# sops -v
sops 3.7.1 (latest)
```

Отлично, с установкой разобрались - теперь давайте сгенерируем приватный pgp ключ и попробуем зашифровать любой текстовый файл:

```
root@client:~# gpg --gen-key
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation,
Inc.
This is free software: you are free to change and redistribute
it.
There is NO WARRANTY, to the extent permitted by law.
```

```
Note: Use "gpg --full-generate-key" for a full featured key
generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Vasiliy Ozerov
Email address: ov@rebrainme.com
You selected this USER-ID:
    "Vasiliy Ozerov <ov@rebrainme.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? O

gpg: key 371D88F68010316D marked as ultimately trusted
gpg: directory '/root/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as
'/root/.gnupg/openpgp-revocs.d/4FEF8890BAB62A70A6FE0F63371D88F68
010316D.rev'
public and secret key created and signed.

pub   rsa3072 2021-04-19 [SC] [expires: 2023-04-19]
      4FEF8890BAB62A70A6FE0F63371D88F68010316D
uid                      Vasiliy Ozerov <ov@rebrainme.com>
sub   rsa3072 2021-04-19 [E] [expires: 2023-04-19]

root@client:~#
```

Как видно мы указали только свое имя и email адрес - они нужны для того чтобы мы могли различать ключи между собой. Давайте посмотрим на все ключи в нашей системе:

```
root@client:~# gpg -k
/root/.gnupg/pubring.kbx
------------------------

pub   rsa3072 2021-04-19 [SC] [expires: 2023-04-19]
      4FEF8890BAB62A70A6FE0F63371D88F68010316D
uid           [ultimate] Vasiliy Ozerov <ov@rebrainme.com>
sub   rsa3072 2021-04-19 [E] [expires: 2023-04-19]

root@client:~#
```

Здесь мы видим публичный идентификатор ключа, который пригодится нам позднее - для шифрования и расшифровки данных - 4FEF8890BAB62A70A6FE0F63371D88F68010316D. Давайте теперь заставим sops создать зашифрованный файл:

```
# sops --pgp 4FEF8890BAB62A70A6FE0F63371D88F68010316D data.yaml
```

Здесь мы указали идентификатор публичного pgp ключа, который будет использован для шифрования данных. После запуска этой команды, sops откроет файл на редактирование в вашем редакторе (который задан переменной EDITOR). Вы можете записать туда любые нужные вам данные. После сохранения мы можем посмотреть на получившийся результат:

```
# cat data.yaml
private_env_variables:
    db_uri:
ENC[AES256_GCM,data:52k0HEPQA7s1LitBdhuPzaQGhQijAtIJwck2xiP7Pms8
/riFsZffBjnTIqrLqPb2QpZ+e0+Zu/HluA==,iv:MkwrbG+hFsJA78V7vHrs6DgJ
h+oply9Zxv+NGx5mD+A=,tag:bdqtiE7Re9E4/wSC+HfW3Q==,type:str]
sops:
    kms: []
    gcp_kms: []
    azure_kv: []
    hc_vault: []
    age: []
    lastmodified: "2021-04-19T16:48:56Z"
    mac:
ENC[AES256_GCM,data:Jml4onNlyUz5Z1t219BSusrYKt7AP/b1RR8jW+KXb0k9
gsc5n1hv7lb0mypxvduVwqDxx9JI7/m2aJdiB+/sWo+iuQJ6OtAJ8ZXzBtyQsoKu
24jPwy0aNIkcmig1QZredhQyoIykWxghpus+cMX3e4vTV8KnZAN3/JIQ77sJs4o=
,iv:6XW/688bYMBC86OqanbyMOpUXXvF+55m16Tep0ELUyA=,tag:+NemAtLgYsx
8tk0twJ4fQg==,type:str]
    pgp:
        - created_at: "2021-04-19T16:48:13Z"
          enc: |
            -----BEGIN PGP MESSAGE-----


hQGMA0qniJIaVvlGAQwAnWUhABYTy3FAUI9xi3k/Emxsp7vvLLxSiPA0hpR5Fynk

spaHrRokYsGO6PDmFT+zyb2VA3DiG+U3afN9DPisOg/FK7R7Ykf4VMe4OZ96CW5W

zF/MEazceHXixzj0FbQ6VU5qZy9ZqvYCTRUa6Xwe4H6d1nZsnQ0rsWSm4fCg9jmv

bN+J9zYh+uBJNbxVJTGqhGWBzYCY2qXmRWnD3ucokVtc2HlyTc7EEYxpMwaqa0l5
```

```
J7nyAt96Dwg/TyIxIKt95kGh5Dpkr6vv5BGeoNrOpA1N3cLfJb5BQOAQTJgKfZC3

BVj7phaWONCnxgET2VWwbEaN4/hiMnSwbELwt7eDB6kFi38eL3V+6sAV1B7xv+0Q

6AwWdrot2T3sxSOhV5VC7prIFnW9msCYu13fjZ4kg9903MngbrfxYwFkzG5PRX5r

jahvziotQlQ4a3QXzrM0AS1j3aOlqXVu82W+eAnNLXWT7Rpm8cYEbaIpdSbfI1lz

buqsLOlv/D3OdebdwgVi0l4B8UWBAjDxmZC8cmB4SW6Xvy9E1NC1lBTe5+Oq30EE

FYxUyfY67SfMGwlcWFhHYI7of+fuRXze2qISKaDWpbl20dos7ZcaQkfxfBlVpz/u
            4dLQY6KyieEM2HWCCpIy
            =RBK8
            -----END PGP MESSAGE-----
          fp: 4FEF8890BAB62A70A6FE0F63371D88F68010316D
    unencrypted_suffix: _unencrypted
    version: 3.7.1
```

Как видно из вывода - наш файл стал нечитаемым. Чтобы прочитать его мы можем указать опцию -d (decrypt):

```
# sops -d data.yaml
private_env_variables:
    db_uri:
postgresql://user:password@1.1.1.1:5432/db?sslmode=disable
```

Как можно заметить внутри зашифрованного файла sops записал public fingerprint, чтобы иметь возможность найти приватный pgp ключ и расшифровать данные (fp: 4FEF8890BAB62A70A6FE0F63371D88F68010316D). Помимо этого, sops указал нам конфигурацию unencrypted_suffix: _unencrypted - благодаря этому суффиксу мы можем указывать переменные в файле, которые sops шифровать не будет. Давайте попробуем открыть файл на редактирование и добавить какую-нибудь переменную, например db_host_unencrypted:

```
# cat data.yaml
private_env_variables:
    db_uri:
ENC[AES256_GCM,data:52k0HEPQA7s1LitBdhuPzaQGhQijAtIJwck2xiP7Pms8
/riFsZffBjnTIqrLqPb2QpZ+e0+Zu/HluA==,iv:MkwrbG+hFsJA78V7vHrs6DgJ
h+oply9Zxv+NGx5mD+A=,tag:bdqtiE7Re9E4/wSC+HfW3Q==,type:str]
db_host_unencrypted: db_host
sops:
    kms: []
```

```
    gcp_kms: []
    azure_kv: []
    hc_vault: []
    age: []
    lastmodified: "2021-04-19T16:51:47Z"
    mac:
ENC[AES256_GCM,data:EvhqF0C4UWmVbE4icd9PlnIYIoNQ3+IHq0fgB8OGswoM
E2gGAbX7cgahrUbAkj7zfkAp7X/2dicCBxvs/d+hOLVsS/H2kCtaQ8lZ/qwE+h5n
F+oozQ3ai+zJ+SAaQ3zdUBvkukWrbr15SlKqTOrvXCEdgFDAi8wVvbemuTFOuAU=
,iv:BQ9HHqqVYkhBPwbE3F9Wj0hOlzLDEE8MQ3V23745C/E=,tag:rnJ4FtbfZMw
3wqCOH9Dkiw==,type:str]
    pgp:
        - created_at: "2021-04-19T16:48:13Z"
          enc: |
            -----BEGIN PGP MESSAGE-----


hQGMA0qniJIaVvlGAQwAnWUhABYTy3FAUI9xi3k/Emxsp7vvLLxSiPA0hpR5Fynk

spaHrRokYsGO6PDmFT+zyb2VA3DiG+U3afN9DPisOg/FK7R7Ykf4VMe4OZ96CW5W

zF/MEazceHXixzj0FbQ6VU5qZy9ZqvYCTRUa6Xwe4H6d1nZsnQ0rsWSm4fCg9jmv

bN+J9zYh+uBJNbxVJTGqhGWBzYCY2qXmRWnD3ucokVtc2HlyTc7EEYxpMwaqa0l5

J7nyAt96Dwg/TyIxIKt95kGh5Dpkr6vv5BGeoNrOpA1N3cLfJb5BQOAQTJgKfZC3

BVj7phaWONCnxgET2VWwbEaN4/hiMnSwbELwt7eDB6kFi38eL3V+6sAV1B7xv+0Q

6AwWdrot2T3sxSOhV5VC7prIFnW9msCYu13fjZ4kg9903MngbrfxYwFkzG5PRX5r

jahvziotQlQ4a3QXzrM0AS1j3aOlqXVu82W+eAnNLXWT7Rpm8cYEbaIpdSbfI1lz

buqsLOlv/D3OdebdwgVi0l4B8UWBAjDxmZC8cmB4SW6Xvy9E1NC1lBTe5+Oq30EE

FYxUyfY67SfMGwlcWFhHYI7of+fuRXze2qISKaDWpbl20dos7ZcaQkfxfBlVpz/u
            4dLQY6KyieEM2HWCCpIy
            =RBK8
            -----END PGP MESSAGE-----
          fp: 4FEF8890BAB62A70A6FE0F63371D88F68010316D
    unencrypted_suffix: _unencrypted
```

```
    version: 3.7.1
root@client:~#
```

Смотрите как интересно - наша переменная db_host_unencrypted: db_host осталась незашифрованной. Эта функция специально добавлена в sops, чтобы вы могли хранить как зашифрованные, так и открытые данные в одном и том же файле.
Итак, с шифрованием разобрались, теперь осталось разобраться с тем как вы можете передать приватный ключ вашему коллеге, чтобы он мог расшифровать ваши данные. Делается это очень просто с использованием gpg:

```
# gpg --armor --export-secret-key
4FEF8890BAB62A70A6FE0F63371D88F68010316D
-----BEGIN PGP PRIVATE KEY BLOCK-----

lQVYBGB9s2gBDADRd7150bQSbkS+c1CsmWoRNyIwafnIt5qmq62qm1LS9c8ofwM7
9msT4s0doJk/OTjL4VEPNPDB56XebzSfIz5qj07HGB32/OpxZ5QODnauaW4gmEOf
7gDW+89y+Py3j4+jX70ozO5gWK2XnBa/2TBxTrqwgyD0Q9HxKbiBsq7CB0rlSNCh

... skipped ...
```

Полученный файл можно передать коллеги и он сможет импортировать его с помощью:

```
# gpg --import private.asc
```

Таким образом вы можете обмениваться приватными файлами и без последствий размещать их в системе контроля версий. Никто, кроме тех у кого есть ключ не смогут их прочитать.
Helm plugins
Отлично, с шифрованием разобрались, теперь самое время разобраться с helm и помочь ему использовать зашифрованные файлы. В этом нам поможет специальный плагин secrets.
По сути, плагины позволяют расширять функциональность хельма и добавлять в него новые функции. Список доступных плагинов можно найти по ссылке.
Для установки плагина можно воспользоваться следующей командой:

```
$ helm plugin install <path|url>
```

В аргументе можно передавать url GitHub repo, которая содержит интересующий вас плагин. Вышеуказанная команда просто клонирует или копирует файлы плагина по пути $XDG_DATA_HOME/plugins. Информацию касательно того, как можно собирать плагины из исходников, можно найти в официальной документации.
Итак, пришло время установить плагин helm secrets.

```
# helm plugin install https://github.com/jkroepke/helm-secrets
Installed plugin: secrets
```

Итак, как же работает данный плагин? На самом деле все очень просто. Вам всего лишь необходимо использовать специальную новую команду secrets в helm. При запуске в этом режиме, secrets плагин запустит sops и расшифрует values файл, а после этого вызовет helm, который будет использовать уже расшифрованную версию values.yaml. Выглядеть это может примерно так (на примере нашего data.yaml):

```
# helm secrets upgrade --install data -f data.yaml ./data
[helm-secrets] Decrypt: data.yaml
... skipped ...
```

Как видно из вывода первой строчкой запустился secrets плагин и расшифровал наш файл data.yaml с помощью sops, после чего запустил helm с параметрами upgrade --install data -f data.yaml ./data.
Таким образом, используя плагин secrets и sops вы можете шифровать свои данные и не бояться закидывать их в систему контроля версий.

Полезные ссылки:

- Helm Docs: plugins

Задание:

1. Склонируйте репозиторий https://gitlab.rebrainme.com/kubernetes-local-for-tasks/helm-secrets.
2. Создайте ветку secrets.
3. Сгенерируйте собственный gpg-ключ.
4. Внутри папки .infra, создайте файл secrets.prod.yaml, который будет содержать список private_env_variables, внутри которого будут указаны переменные (переменные надо взять из values.prod.yaml):
- API_KEY
- PGSQL_URI
5. Удалите переменные API_KEY / PGSQL_URI из values.prod.yaml.
6. В итоге у вас должно получиться два файла - values.prod.yaml, который содержит public_env_variables (LISTEN) и secrets.prod yaml, который содержит private_env_variables (API_KEY / PGSQL_URI).
7. Зашифруйте secrets.prod.yaml с помощью sops и ключа, сгенерированного в пункте 3.
8. Создайте репозиторий в группе kubernetes_users_repos/your_gitlab_id/ и запушьте ваши изменения в него.