Настройка сущностей и групп удостоверений

StarVault поддерживает несколько методов аутентификации и позволяет включать один и тот же метод аутентификации на разных путях монтирования. Каждый клиент StarVault может иметь несколько учетных записей в различных провайдерах удостоверений, активированных на сервере StarVault.

Клиенты StarVault могут быть сопоставлены с сущностями, а их соответствующие учетные записи в провайдерах аутентификации — с псевдонимами. По сути, каждая сущность состоит из нуля или более псевдонимов. Движок секретов удостоверений внутренне управляет клиентами, которые распознаются StarVault.

1. Учетные записи

Шаги, описанные в данном руководстве, выполняются под учетной записью оператора.

2. Задача

Боб имеет учетные записи как в userpass и LDAP. Оба метода аутентификации (userpass и LDAP) активированы на сервере StarVault, и он может авторизоваться, используя любую из своих учетных записей. Обе учетные записи принадлежат Бобу, однако между ними нет связи. Из-за этого нет возможности задать для них общие свойства.

3. Решение

Создайте сущность, представляющую Боба, и свяжите с ней псевдонимы, соответствующие каждой из его учетных записей. Можно установить дополнительные политики и метаданные на уровне сущности, чтобы обе учетные записи наследовали их.

Когда Боб авторизуется с использованием любой из своих учетных записей, идентификатор сущности будет привязан к аутентификационному токену. При использовании таких токенов их идентификаторы сущностей фиксируются в журнале аудита, создавая запись действий, выполненных конкретными пользователями.

4. Предварительные требования

Для выполнения описанных в этом руководстве задач вам нужна среда StarVault.

- Обратитесь к руководству по установке StarVault для установки StarVault.
- jq для обработки вывода JSON для удобочитаемости.

5. Требования к политике

Если вы не используете StarVault для тестирования или разработки (например, в dev-режиме -dev), рекомендуется аутентифицироваться в StarVault и получать токены с соответствующим набором политик в зависимости от вашей роли в организации.

Если необходимо подключиться к неразработческому серверу StarVault, ваш токен должен содержать политику с следующими разрешениями:

HCL

```
# Настройка методов аутентификации
path "sys/auth" {
  capabilities = [ "read", "list" ]
}

# Полный доступ к настройке методов аутентификации
path "sys/auth/*" {
  capabilities = [ "create", "update", "read", "delete", "list", "sudo" ]
}

# Управление методом аутентификации userpass
path "auth/userpass/*" {
  capabilities = [ "create", "read", "update", "delete" ]
}

# Просмотр вкладки "Политики" в UI
path "sys/policies" {
  capabilities = [ "read", "list" ]
}

# Создание и управление ACL-политиками через UI
path "sys/policies/acl/*" {
  capabilities = [ "create", "read", "update", "delete", "list" ]
}

# Управление политиками ACL
path "sys/policies/acl" {
  capabilities = [ "read", "list" ]
}

# Полный доступ к управлению политиками ACL
path "sys/policies/acl/*" {
  capabilities = [ "create", "read", "update", "delete", "list" ]
}
```

```
# Получение списка доступных движков секретов для извлечения accessor ID
path "sys/mounts" {
  capabilities = [ "read" ]
}

# Создание и управление сущностями и группами
path "identity/*" {
  capabilities = [ "create", "read", "update", "delete", "list" ]
}
```

Если вы не знакомы с политиками, рекомендуется к прочтению руководство по политикам.

6. Настройка среды

1. В терминале установите переменную окружения `STARVAULT_ADDR` адресом StarVault:

```
BASH
$ export STARVAULT_ADDR=<Public_Cluster_URL>
```

2. В терминале установите переменную окружения `STARVAULT_TOKEN`:

```
BASH
$ export STARVAULT_TOKEN=<token>
```

3. Проверьте подключение к кластеру StarVault:

```
BASH
$ starvault status

Key                     Value
---                     -----
Recovery Seal Type      shamir
Initialized             true
Sealed                  false
Total Recovery Shares   1
Threshold               1
Version                 1.9.2+ent
Storage Type            raft
```

4. Включите движок хранения секретов Key/Value v2 в пути secret:

```
BASH
starvault secrets enable -path=secret kv-v2
```

Теперь сервер StarVault готов к использованию.

7. Создание сущности с псевдонимом

Вам необходимо создать новую сущность с назначенной базовой политикой. Сущность определяет два псевдонима сущности, для каждого из которых назначена своя политика.

Пример сценария: У пользователя Боба Смита из ACME Inc. есть два набора учетных данных: bob и bsmith. Боб может проходить аутентификацию в StarVault, используя любую из его учетных записей. Чтобы управлять учетными записями пользователя и связать их с личностью Боба Смита в команде QA, вам нужно создать объект для Боба.

Для упрощения этого руководства вы будете работать с методом аутентификации `userpass`. В реальности пользователь bob может быть именем пользователя, которое существует в Active Directory, а bsmith может быть именем пользователя Bob на LDAP. Чтобы сымитировать такое поведение, вам нужно включить метод `userpass auth` в двух разных путях: `userpass-test` и `userpass-qa`, чтобы сымитировать два разных типа методов аутентификации.

Блокировка пользователей. В StarVault функция блокировки пользователей включена по умолчанию для методов аутентификации `userpass`, `approle` и `ldap`.

7.1. Политика сценариев

base.hcl

```hcl
path "secret/data/training_*" {
   capabilities = ["create", "read"]
}
```

test.hcl

```hcl
path "secret/data/test" {
   capabilities = [ "create", "read", "update", "delete" ]
}
```

team-qa.hcl

```hcl
path "secret/data/team-qa" {
   capabilities = [ "create", "read", "update", "delete" ]
}
```

В этом сценарии предполагается, что механизм секретов K/V v2 включен на пути секретов. Если вы не уверены, обратитесь к руководству по версионированному механизму секретов ключей/значений.

Создайте пользователей `bob` и `bsmith` с соответствующими политиками.

7.2. CLI команда

1. Создайте политику `base`

```bash
$ starvault policy write base -<<EOF
path "secret/data/training_*" {
   capabilities = ["create", "read"]
}
EOF
```

2. Создайте политику `test`

```bash
$ starvault policy write test -<<EOF
path "secret/data/test" {
   capabilities = [ "create", "read", "update", "delete" ]
}
EOF
```

3. Создайте политику `team-qa`

```bash
$ starvault policy write team-qa -<<EOF
path "secret/data/team-qa" {
   capabilities = [ "create", "read", "update", "delete" ]
}
EOF
```

4. Выведите все политики, чтобы убедиться в существовании политик `base`, `test`, `team-qa`

```bash
$ starvault policy list
base
default
team-qa
test
root
```

5. Включите метод аутентификации `userpass` на `userpass-test`.

```bash
$ starvault auth enable -path="userpass-test" userpass
```

6. Создайте нового пользователя `bob` в `userpass-test`, где пароль - `training`, а политика `test` подключена.

```bash
$ starvault write auth/userpass-test/users/bob password="training"
policies="test"
```

7. Включите метод аутентификации `userpass` на другом пути, `userpass-qa` .

```
$ starvault auth enable -path="userpass-qa" userpass
```

8. Создайте нового пользователя с именем `bsmith` в `userpass-qa` , где пароль - `training` и политика `team-qa` подключена.

```
$ starvault write auth/userpass-qa/users/bsmith password="training"
policies="team-qa"
```

9. Выполните следующую команду, чтобы обнаружить свойство монтирования для метода аутентификации `userpass` .

```
$ starvault auth list -detailed
```

Path	Plugin	Accessor	...
----	----	----	...
`token/`	`token`	`auth_token_c5943123`	...
`userpass-qa/`	`userpass`	`auth_userpass_8c7b8e0f`	...
`userpass-test/`	`userpass`	`auth_userpass_264d4705`	...

...

Каждый метод аутентификации userpass имеет уникальное значение **Accessor (указателя)** для его идентификации.

10. Выполните следующую команду, чтобы сохранить значение указателя `userpass-test auth` в файле с именем `accessor_test.txt` .

```
$ starvault auth list -format=json | jq -r '.["userpass-test/"].accessor' >
accessor_test.txt
```

Полученный файл содержит значение указателя(`auth_userpass_XXXXX`).

11. Аналогичным образом выполните следующую команду, чтобы сохранить значение указателя `userpass-qa` в файле с именем `accessor_qa.txt` .

```
$ starvault auth list -format=json | jq -r '.["userpass-qa/"].accessor' >
accessor_qa.txt
```

12. Создайте сущность для `bob-smith` и сохраните полученный идентификатор сущности в файле с именем `entity_id.txt` .

```
$ starvault write -format=json identity/entity name="bob-smith"
policies="base" \
     metadata=organization="ACME Inc." \
     metadata=team="QA" \
     | jq -r ".data.id" > entity_id.txt
```

Полученный файл содержит идентификатор сущности `bob-smith` (например, `24204b50-22a6-61f5-bd4b-803f1a4e4726`).

13. Теперь добавьте пользователя `bob` к сущности `bob-smith`, создав псевдоним сущности. Установите пользовательские метаданные для псевдонима сущности `bob` с именем `account` и задайте для него значение `Tester Account`.

```
$ starvault write identity/entity-alias name="bob" \
     canonical_id=$(cat entity_id.txt) \
     mount_accessor=$(cat accessor_test.txt) \
     custom_metadata=account="Tester Account"
```

Пример вывода:

Key	Value
`canonical_id`	`24204b50-22a6-61f5-bd4b-803f1a4e4726`
`id`	`ae2cdd0f-9807-7336-2265-5575c71837e7`

Просмотр псевдонимов сущностей (необязательно)

Чтобы просмотреть созданный псевдоним сущности, используйте возвращаемый идентификатор псевдонима сущности.

```
$ starvault read identity/entity-alias/id/<id>
```

Пример

```
$ starvault read identity/entity-alias/id/ae2cdd0f-9807-7336-2265-
5575c71837e7
```

Key	Value
`canonical_id`	`971b5ab3-1931-8381-8605-1b9511572278`
`creation_time`	`2021-11-15T01:41:03.093366Z`

Key	Value
`custom_metadata`	`map[account:Tester Account]`
`id`	`10340e8f-64fa-fe7d-0c88-b6172293f788`
`last_update_time`	`2021-11-15T01:41:03.093366Z`
`local`	`false`
`merged_from_canonical_ids`	`<nil>`
`metadata`	`<nil>`
`mount_accessor`	`auth_userpass_2cbfe431`
`mount_path`	`auth/userpass-test/`
`mount_type`	`userpass`
`name`	`bob`
`namespace_id`	`root`

14. Повторите предыдущий шаг, чтобы добавить пользователя `bsmith` в сущность `bob-smith`. Установите пользовательские метаданные для псевдонима сущности `bob` с именем "account" и задайте для него значение "QA Eng Account".

```
$ starvault write identity/entity-alias name="bsmith" \
     canonical_id=$(cat entity_id.txt) \
     mount_accessor=$(cat accessor_qa.txt) \
     custom_metadata=account="QA Eng Account"
```

Пример вывода:

Key	Value
---	-----
`canonical_id`	`24204b50-22a6-61f5-bd4b-803f1a4e4726`
`id`	`6066f6af-bb1c-5310-58a1-fd9c8f151573`

15. Посмотрите сведения о сущности

```
$ starvault read -format=json identity/entity/id/$(cat entity_id.txt) | jq -
r ".data"
```

Пример вывода: Результат должен включать псевдонимы сущностей, метаданные (организация и команда) и базовую политику.

BASH


```
{
  "aliases": [
    {
      "canonical_id": "73503625-abcd-db22-08c3-c121d682d550",
      "creation_time": "2021-11-17T05:33:48.040506Z",
      "custom_metadata":
      {
        "account": "Tester Account"
      },
      "id": "cf073e2e-41af-852f-848d-f67533c8a610",
      "last_update_time": "2021-11-17T05:33:48.040506Z",
      "local": false,
      "merged_from_canonical_ids": null,
      "metadata": null,
      "mount_accessor": "auth_userpass_0a6936a7",
      "mount_path": "auth/userpass-test/",
      "mount_type": "userpass",
      "name": "bob"
    },
     {
      "canonical_id": "73503625-abcd-db22-08c3-c121d682d550",
      "creation_time": "2021-11-17T05:33:48.107834Z",
      "custom_metadata":
      {
        "account": "QA Eng Account"
      },
      "id": "7add6763-ce53-d92a-c795-c8ae529ce6e7",
      "last_update_time": "2021-11-17T05:33:48.107834Z",
      "local": false,
      "merged_from_canonical_ids": null,
      "metadata": null,
      "mount_accessor": "auth_userpass_ef4f8068",
      "mount_path": "auth/userpass-qa/",
      "mount_type": "userpass",
      "name": "bsmith"
     }
     ],
  "creation_time": "2021-11-17T05:33:47.966585Z",
  "direct_group_ids": [
    "49e8b9e8-8933-1e14-f05c-e1c9674b142b"
  ],
  "disabled": false,
  "group_ids": [
    "49e8b9e8-8933-1e14-f05c-e1c9674b142b"
  ],
  "id": "73503625-abcd-db22-08c3-c121d682d550",
  "inherited_group_ids": [],
  "last_update_time": "2021-11-17T05:33:47.966585Z",
```

```
  "merged_entity_ids": null,
  "metadata":
  {
    "organization": "ACME Inc.",
    "team": "QA"
  },
  "mfa_secrets": {},
  "name": "bob-smith",
  "namespace_id": "root",
  "policies":
  [
    "base"
  ]
}
```

7.3. Вызов API с использованием cURL

1. Сначала создайте полезную нагрузку запроса API, содержащую базовую политику.

```
$ tee payload-1.json <<EOF
{
  "policy": "path \"secret/data/training_*\" {\n   capabilities =
[\"create\", \"read\"]\n}"
}
EOF
```

2. Создайте политику `base`.

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request PUT \
   --data @payload-1.json \
   $STARVAULT_ADDR/v1/sys/policies/acl/base
```

3. Создайте полезную нагрузку API запроса, содержащую потилику `test`.

```
$ tee payload-2.json <<EOF
{
  "policy": "path \"secret/data/test\" {\n capabilities = [ \"create\",
\"read\", \"update\", \"delete\" ]\n }"
}
EOF
```

4. Создайте политику `test`

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request PUT \
```

```
   --data @payload-2.json \
   $STARVAULT_ADDR/v1/sys/policies/acl/test
```

5. Создайте полезную нагрузку запроса API, содержащую политику `team-qa`.

```
$ tee payload-3.json <<EOF
{
  "policy": "path \"secret/data/team-qa\" {\n capabilities = [ \"create\", \"read\", \"update\", \"delete\" ]\n }"
}
EOF
```

6. Создайте политику `team-qa`

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request PUT \
   --data @payload-3.json \
   $STARVAULT_ADDR/v1/sys/policies/acl/team-qa
```

7. Выведите все политики, чтобы убедиться в существовании политик `base`, `test` и `team-qa`.

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request LIST \
   $STARVAULT_ADDR/v1/sys/policies/acl | jq -r ".data"
```

Вывод:

```
{
  "keys": [
    "base",
    "default",
    "team-qa",
    "test",
    "root"
  ]
}
```

8. Включите метод аутентификации `userpass` на `userpass-test`.

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data '{"type": "userpass"}' \
   $STARVAULT_ADDR/v1/sys/auth/userpass-test
```

9. Создайте нового пользователя `bob` в `userpass-test`, где пароль - `training`, а политика `test` подключена.

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data '{"password": "training", "policies": "test"}' \
   $STARVAULT_ADDR/v1/auth/userpass-test/users/bob
```

10. Включите метод аутентификации `userpass` в `userpass-qa`.

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data '{"type": "userpass"}' \
   $STARVAULT_ADDR/v1/sys/auth/userpass-qa
```

11. Создайте нового пользователя с именем `bsmith` в `userpass-qa`, где пароль - `training`, а политика `team-qa` подключена.

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data '{"password": "training", "policies": "team-qa"}' \
   $STARVAULT_ADDR/v1/auth/userpass-qa/users/bsmith
```

12. Выполните следующую команду, чтобы обнаружить ссылку монтирования для метода аутентификации `userpass`, включенного в `userpass-test`, и сохраните его значение в файле 'accessor_test.txt'

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   $STARVAULT_ADDR/v1/sys/auth | jq -r '.data | .["userpass-
test/"].accessor' > accessor_test.txt
```

    Результирующий файл содержит значение указателя (`auth_userpass_XXXXX`).

13. Выполните следующую команду, чтобы обнаружить указатель монтирования для метода аутентификации `userpass`, включенного в `userpass-qa`, и сохраните его значение в файле `accessor_qa.txt`.

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   $STARVAULT_ADDR/v1/sys/auth | jq -r '.data | .["userpass-qa/"].accessor'
> accessor_qa.txt
```

14. Создайте полезную нагрузку API для создания сущности `bob-smith`.

```bash
$ tee payload-entity.json <<EOF
{
  "name": "bob-smith",
  "metadata": {
    "organization": "ACME Inc.",
    "team": "QA"
  },
```

```
  "policies": ["base"]
}
EOF
```

15. Создайте сущность `bob-smith`

```
BASH
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data @payload-entity.json \
   $STARVAULT_ADDR/v1/identity/entity | jq -r ".data.id" > entity_id.txt
```

Полученный файл содержит идентификатор сущности `bob-smith` (например, `24204b50-22a6-61f5-bd4b-803f1a4e4726`).

16. Добавьте пользователя `bob` к сущности `bob-smith`, создав псевдоним сущности. В теле запроса нужно передать имя userpass как `name`, значение указателя `userpass-test` как `mount_accessor` и id сущности как `canonical_id`. Установите пользовательские метаданные для псевдонима сущности `bob` с именем "account" и задайте его значение как "Tester Account".

Создайте полезную API нагрузку.

```
BASH
$ tee payload-bob.json <<EOF
{
  "name": "bob",
  "canonical_id": "$(cat entity_id.txt)",
  "mount_accessor": "$(cat accessor_test.txt)",
  "custom_metadata": {
    "account": "Tester Account"
  }
}
EOF
```

Используйте конечную точку `identity/entity-alias`, чтобы добавить псевдоним сущности.

```
BASH
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data @payload-bob.json \
   $STARVAULT_ADDR/v1/identity/entity-alias | jq -r ".data"
```

Пример вывода:

```
BASH
{
  "canonical_id": "24204b50-22a6-61f5-bd4b-803f1a4e4726",
  "id": "ae2cdd0f-9807-7336-2265-5575c71837e7"
}
```

Просмотр псевдонимов сущностей (необязательно)

Чтобы просмотреть созданный псевдоним сущности, используйте возвращаемый идентификатор псевдонима сущности.

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   $STARVAULT_ADDR/v1/identity/entity-alias/id/<id> | jq -r ".data"
```

Пример

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   $STARVAULT_ADDR/v1/identity/entity-alias/id/ae2cdd0f-9807-7336-2265-
5575c71837e7 \
   | jq -r ".data"

{
  "canonical_id": "65764de8-4e3a-6ea1-8c96-3df951f05b42",
  "creation_time": "2021-11-15T02:18:18.925287Z",
  "custom_metadata":
  {
    "account": "Tester Account"
  },
  "id": "ad67d5f0-221f-5c4a-b652-eecf107e5beb",
  "last_update_time": "2021-11-15T02:18:18.925287Z",
  "local": false,
  "merged_from_canonical_ids": null,
  "metadata": null,
  "mount_accessor": "auth_userpass_53c9a0c7",
  "mount_path": "auth/userpass-test/",
  "mount_type": "userpass",
  "name": "bob",
  "namespace_id": "root"
}
```

17. Повторите предыдущий шаг, чтобы добавить пользователя `bsmith` в сущность `bob-smith`. Установите пользовательские метаданные для псевдонима сущности `bob` с именем "account" и задайте для него значение "QA Eng Account".

    Создайте полезную нагрузку API-запроса.

```
$ tee payload-bsmith.json <<EOF
{
   "name": "bsmith",
   "canonical_id": "$(cat entity_id.txt)",
   "mount_accessor": "$(cat accessor_qa.txt)",
   "custom_metadata": {
     "account": "QA Eng Account"
   }
```

```
}
EOF
```

Используйте конечную точку `identity/entity-alias`, чтобы добавить псевдоним сущности.

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data @payload-bsmith.json \
   $STARVAULT_ADDR/v1/identity/entity-alias | jq -r ".data"
```

Пример вывода:

```
{
  "canonical_id": "24204b50-22a6-61f5-bd4b-803f1a4e4726",
  "id": "6066f6af-bb1c-5310-58a1-fd9c8f151573"
}
```

18. Просмотрите сведения о сущности

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   $STARVAULT_ADDR/v1/identity/entity/id/$(cat entity_id.txt) | jq -r
".data"
```

Пример вывода: Пользователи `bob` и `bsmith` должны появиться в списке псевдонимов сущности и в метаданных, связанных с ней.

```
{
  "aliases": [
    {
      "canonical_id": "73503625-abcd-db22-08c3-c121d682d550",
      "creation_time": "2021-11-17T05:33:48.040506Z",
      "custom_metadata":
      {
        "account": "Tester Account"
      },
      "id": "cf073e2e-41af-852f-848d-f67533c8a610",
      "last_update_time": "2021-11-17T05:33:48.040506Z",
      "local": false,
      "merged_from_canonical_ids": null,
      "metadata": null,
      "mount_accessor": "auth_userpass_0a6936a7",
      "mount_path": "auth/userpass-test/",
      "mount_type": "userpass",
      "name": "bob"
    },
    {
      "canonical_id": "73503625-abcd-db22-08c3-c121d682d550",
      "creation_time": "2021-11-17T05:33:48.107834Z",
```

```
      "custom_metadata":
      {
        "account": "QA Eng Account"
      },
      "id": "7add6763-ce53-d92a-c795-c8ae529ce6e7",
      "last_update_time": "2021-11-17T05:33:48.107834Z",
      "local": false,
      "merged_from_canonical_ids": null,
      "metadata": null,
      "mount_accessor": "auth_userpass_ef4f8068",
      "mount_path": "auth/userpass-qa/",
      "mount_type": "userpass",
      "name": "bsmith"
    }
  ],
  "creation_time": "2021-11-17T05:33:47.966585Z",
  "direct_group_ids": [
    "49e8b9e8-8933-1e14-f05c-e1c9674b142b"
  ],
  "disabled": false,
  "group_ids": [
    "49e8b9e8-8933-1e14-f05c-e1c9674b142b"
  ],
  "id": "73503625-abcd-db22-08c3-c121d682d550",
  "inherited_group_ids": [],
  "last_update_time": "2021-11-17T05:33:47.966585Z",
  "merged_entity_ids": null,
  "metadata":
  {
    "organization": "ACME Inc.",
    "team": "QA"
  },
  "mfa_secrets": {},
  "name": "bob-smith",
  "namespace_id": "root",
  "policies": [
    "base"
  ]
}
```


7.4. Веб-интерфейс

1. Откройте веб-браузер и запустите StarVault UI с добавлением `http://127.0.0.1:8200/ui` в адрес и войдите в систему.
2. Перейдите на вкладку **Policies** и выберите **Create ACL policy** .
3. Введите `base` в поле **Name** и вставьте следующую политику в текстовое поле **Policy**.

```
path "secret/data/training_*"
{
   capabilities = ["create", "read"]
}
```

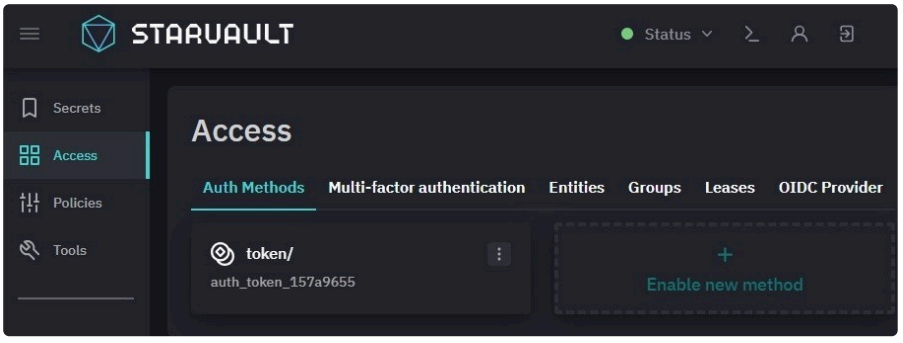
4. Нажмите кнопку **Create Policy** для завершения
5. Снова выберите **Create ACL policy**, введите `test` в поле **Name** и вставьте следующую политику в текстовое поле **Policy**.

```
path "secret/data/test"
{
   capabilities = [ "create", "read", "update", "delete" ]
}
```

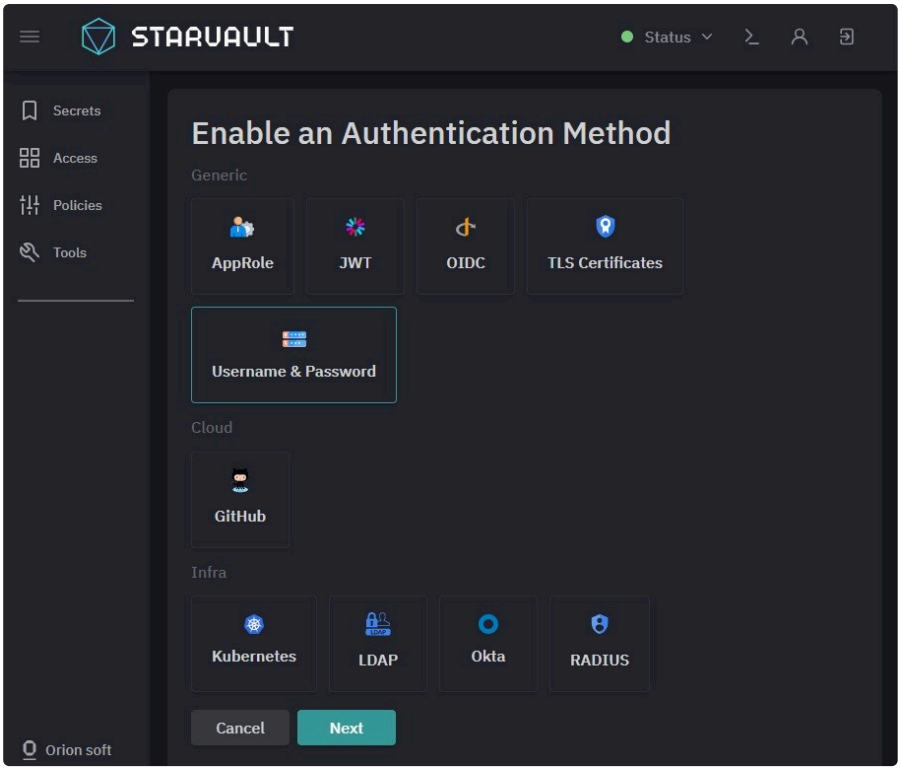
6. Нажмите кнопку **Create Policy** для завершения.
7. Снова выберите **Create ACL policy**, введите `team-qa` в поле **Name** и вставьте следующую политику в текстовое поле **Policy**.

```
path "secret/data/team-qa"
{
   capabilities = [ "create", "read", "update", "delete" ]
}
```

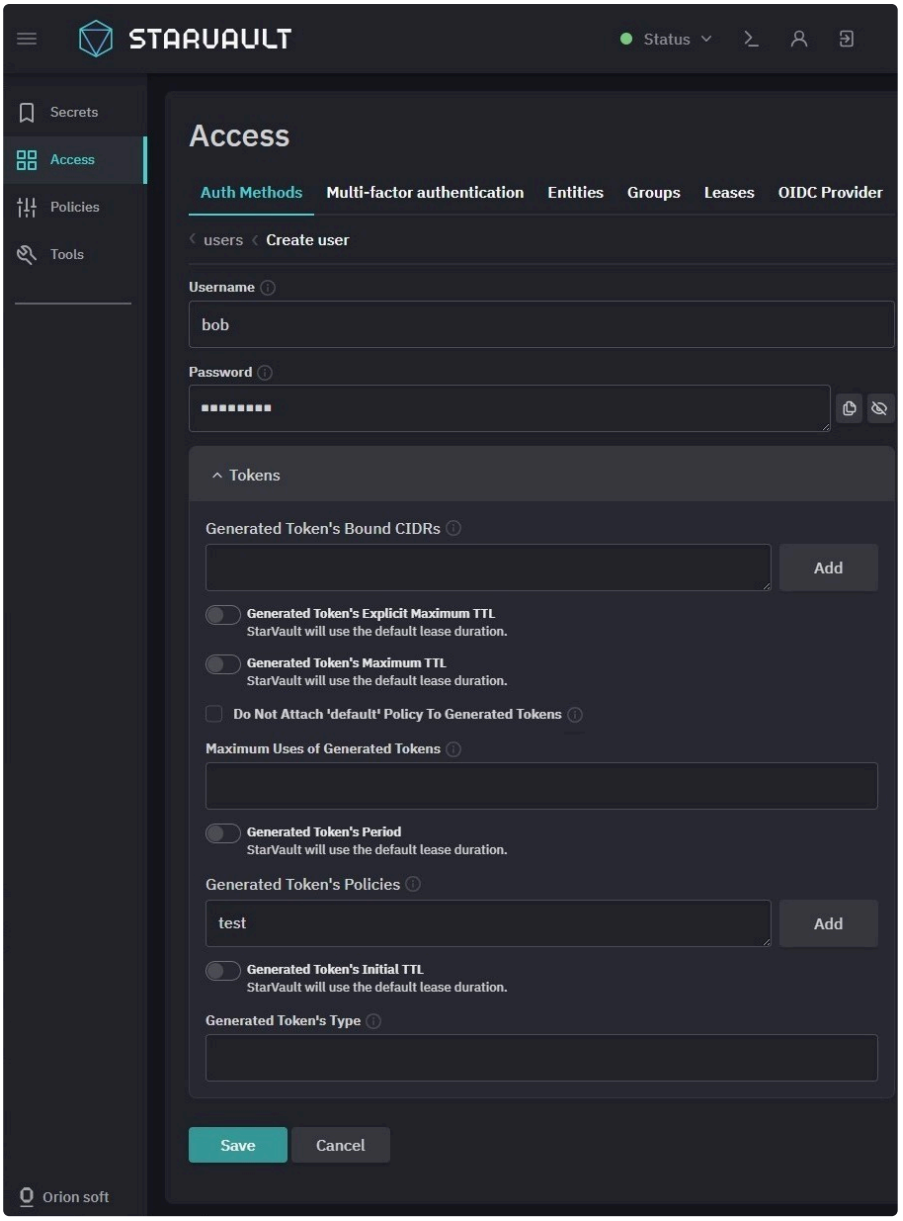
8. Нажмите кнопку **Create Policy** для завершения.
9. Перейдите на вкладку **Access** и выберите **Enable new method**.



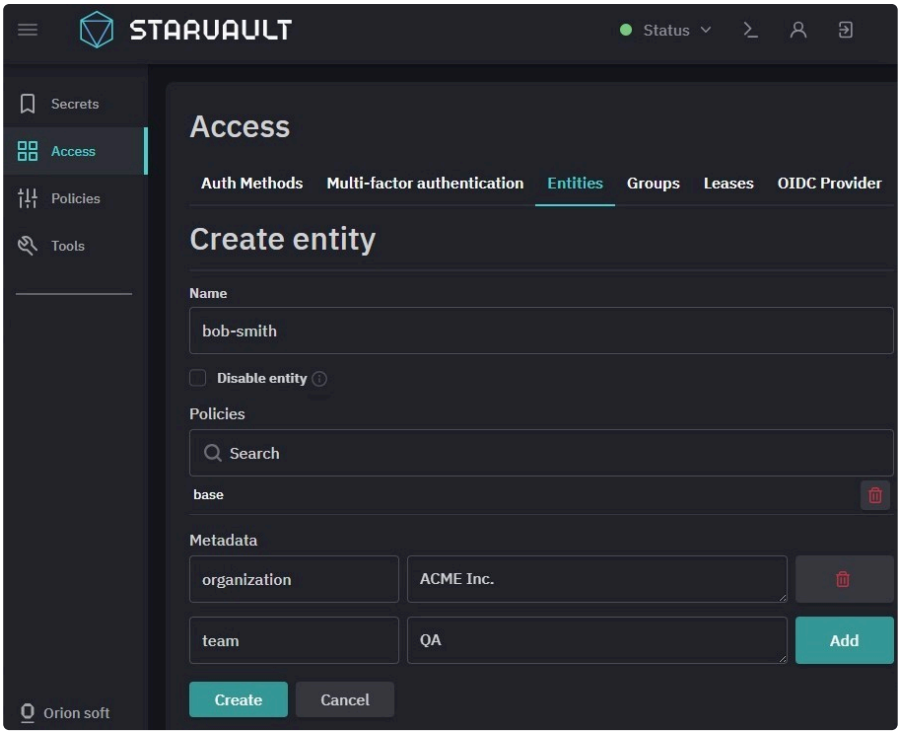
10. Нажмите **Username & Password**.



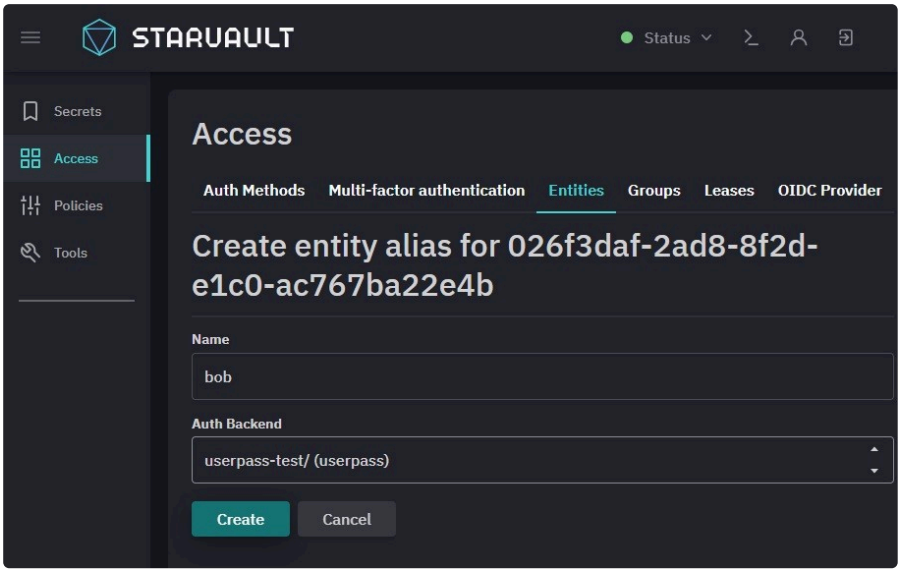
11. Нажмите **Next**
12. Введите `userpass-test` в поле **Path** и нажмите **Enable Method**.
13. Снова перейдите на вкладку **Access**, а затем выберите **userpass-test/**.
14. Выберите **Create user**. Введите `bob` в поле **Username** и `training` в поле **Password**
15. Разверните вкладку **Tokens**
16. В разделе **Generated Token's Policies** введите `test`, чтобы прикрепить политику тестирования



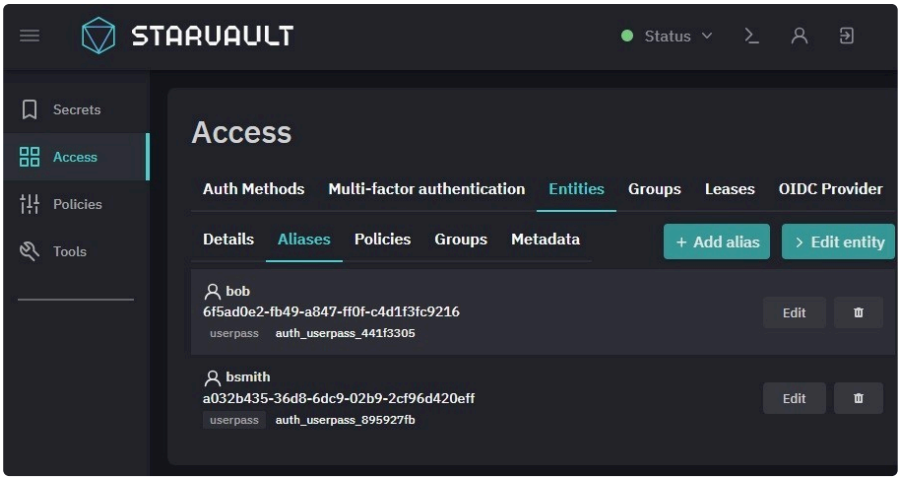
17. Нажмите **Save**
18. Выберите **Auth Methods** и **Enable new method**.
19. Выберите **Username & Password**, затем нажмите **Next**
20. Введите `userpass-qa` в поле **Path** и нажмите **Enable Method**.
21. Снова нажмите на вкладку **Access** и выберите **userpass-qa/**.
22. Выберете **Create user**. Введите `bsmith` в поле **Username** и `training` в поле **Password**.
23. Разверните вкладку **Tokens**
24. В разделе **Generated Token's Policies** введите `team-qa`, чтобы прикрепить политику тестирования
25. Нажмите **Save**.
26. На вкладке **Access** выберите **Create entity**, чтобы создать новую сущность `bob-smith`. Заполните поля **Name**, **Policies** и **Metadata**, как показано ниже:



27. Нажмите **Create**.
28. Выберите **Add alias**. Введите `bob` в поле **Name** и выберите `userpass-test/ (userpass)` в раскрывающемся списке **Auth Backend**



29. Нажмите **Create**.
30. Вернитесь в список **Entities**. Выберите **Add alias** в меню сущности `bob-smith`, затем **Add alias**.
31. Введите `bsmith` в поле **Name** и выберите `userpass-qa/ (userpass)` в раскрывающемся списке **Auth Backend**, затем нажмите **Create**
32. Вернитесь в раздел **Entities > bob-smith** и выберите **Aliases**. В этом списке должны быть `bob` и `bob-smith`



Соображения безопасности. Избегайте хранения в метаданных сущности какой-либо конфиденциальной персональной информации (PII). Метаданные сущности реплицируются на другие кластеры, если настроен Performance Replication. Это может стать серьезной проблемой в связи с GDPR. В StarVault есть возможность устанавливать пользовательские метаданные для каждого псевдонима сущности, которые не пересекаются с метаданными, установленными StarVault. Если метод аутентификации является локальным для кластера, метаданные не будут реплицироваться на другие кластеры в той же группе репликации производительности. Поэтому рекомендуется использовать пользовательские метаданные для псевдонима сущности.

8. Тестирование сущности

Чтобы лучше понять, как токен наследует возможности от политики сущности, можно протестировать это, войдя в систему под именем bob.

8.1. CLI команда

1. Войдите в систему под именем `bob` и сохраните сгенерированный клиентский токен в файле `bob_token.txt`.

```bash
$ starvault login -format=json -method=userpass -path=userpass-test \
    username=bob password=training \
    | jq -r ".auth.client_token" > bob_token.txt
```

2. Политика `test` разрешает CRUD-операции на пути `secret/test`. Убедитесь, можете ли вы записывать секреты в этот путь.

```bash
$ STARVAULT_TOKEN=$(cat bob_token.txt) starvault kv put secret/test
owner="bob"
```

Key	Value
`created_time`	`2021-11-06T02:12:02.104146Z`
`custom_metadata`	`<nil>`
`deletion_time`	`n/a`
`destroyed`	`false`
`destroyed`	`false`
`version`	`1`

3. Хотя имя пользователя `bob` не имеет подключенной `base` политики, токен наследует возможности, предоставленные базовой политикой, потому что `bob` является членом сущности `bob-smith`, а эта сущность имеет подключенную базовую политику. Проверьте, унаследовал ли `bob` эти возможности.

```
$ STARVAULT_TOKEN=$(cat bob_token.txt) starvault token capabilities
secret/data/training_test
create, read
```

4. Базовая политика предоставляет возможности создания и чтения для пути `secret/training_*`; поэтому Бобу разрешено выполнять операции создания и чтения для любого пути, начинающегося с `secret/training_*`.

5. А как насчет пути `secret/team-qa`?

```
$ STARVAULT_TOKEN=$(cat bob_token.txt) starvault token capabilities
secret/data/team-qa
deny
```

Пользователь `bob` наследует возможности только от политики ассоциированной с ним сущности. Пользователь может получить доступ к пути `secret/team-qa`, только если он вошел в систему с учетными данными `bsmith`. ---

8.2. Вызов API с использованием cURL

1. Войдите в систему под именем `bob` и сохраните сгенерированный клиентский токен в файле `bob_token.txt`.

```
$ curl --request POST \
   --data '{"password": "training"}' \
```

```
   $STARVAULT_ADDR/v1/auth/userpass-test/login/bob \
   | jq -r ".auth.client_token" > bob_token.txt
```

2. Политика `test` разрешает CRUD-операции на пути `secret/test`. Убедитесь, можете ли вы записывать секреты в этот путь.

   Создайте полезную нагрузку API-запроса.

```
BASH
tee payload_test.json <<EOF
{
  "data": {
    "owner": "bob"
  }
}
EOF
```

3. Проверьте, можно ли записывать секреты по пути

```
BASH
curl --header "X-Vault-Token: $(cat bob_token.txt)" \
   --request POST \
   --data @payload_test.json \
   $STARVAULT_ADDR/v1/secret/data/test | jq -r ".data"
```

   Команда выше должна быть выполнена успешно.

   Пример вывода

```
BASH
{
  "created_time": "2021-11-06T02:31:57.43338Z",
  "custom_metadata": null,
  "deletion_time": "",
  "destroyed": false,
  "version": 1
}
```

4. Хотя имя пользователя `bob` не имеет подключенной политики `base`, токен наследует возможности, предоставленные базовой политикой, потому что `bob` является членом сущности `bob-smith`, а эта сущность имеет подключенную базовую политику. Проверьте, унаследовал ли `bob` эти возможности.

```
BASH
$ curl --header "X-Vault-Token: $(cat bob_token.txt)" \
   --request POST \
   --data '{"paths": ["secret/data/training_test"]}' \
   $STARVAULT_ADDR/v1/sys/capabilities-self | jq -r ".data"
```

   Пример вывода:

```
{
  "capabilities": [
    "create",
    "read"
  ],
  "secret/data/training_test": [
    "create",
    "read"
  ]
}
```

Политика `base` предоставляет возможности создания и чтения для пути `secret/training_*`; поэтому `bob` разрешено выполнять операции создания и чтения для любого пути, начинающегося с `secret/training_*`.

5. А как насчет пути `secret/team-qa`?

```
$ curl --header "X-Vault-Token: $(cat bob_token.txt)" \
   --request POST \
   --data '{"paths": ["secret/data/team-qa"]}' \
   $STARVAULT_ADDR/v1/sys/capabilities-self | jq -r ".data"
```

Пример вывода:

```
{
  "capabilities": [
    "deny"
  ],
  "secret/data/team-qa": [
    "deny"
  ]
}
```

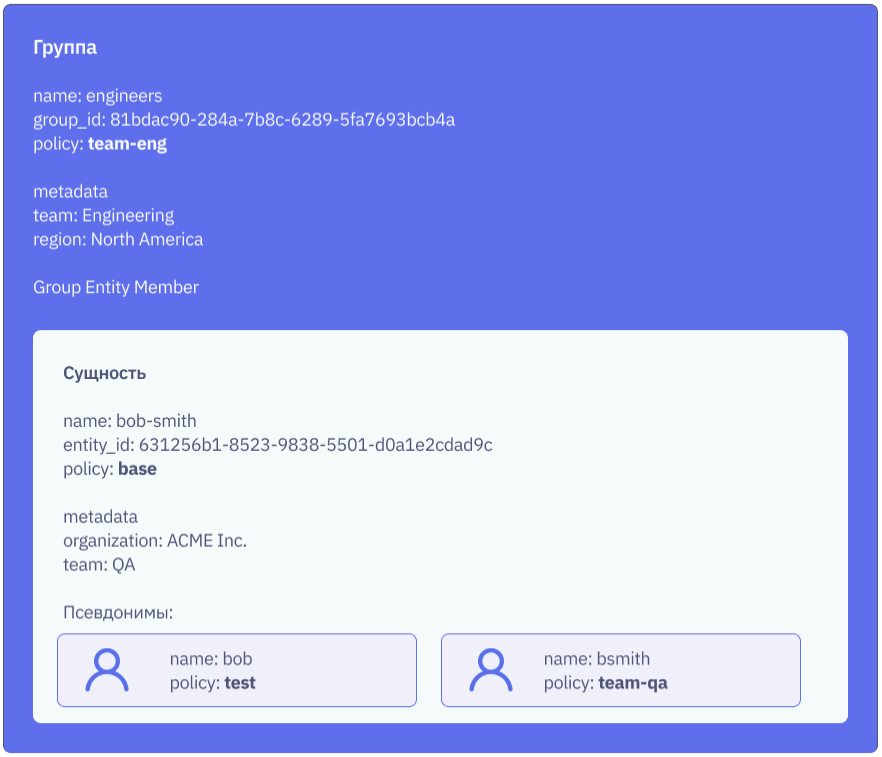
Пользователь `bob` наследует возможности только от политики ассоциированной с ним сущности. Пользователь может получить доступ к пути `secret/team-qa`, только если он вошел в систему с учетными данными `bsmith`.

9. Создание внутренней группы

Для создания внутренней группы сбросьте переменную окружения **STARVAULT_TOKEN**

Войдите в систему с токеном, который вы использовали при настройке сущности, прежде чем приступить к созданию внутренней группы. (См. раздел Настройка среды).

Пусть необходимо создать внутреннюю группу под названием `engineers`. Ее членом будет сущность `bob-smith`, которая была создана в разделе Создание сущности с псевдонимом.



Групповая политика `team-eng` включает следующее в файле `team-eng.hcl.`.

team-eng.hcl

```hcl
path "secret/data/team/eng" {
  capabilities = [ "create", "read", "update", "delete"]
}
```

9.1. CLI команда

1. Создайте новую политику `team-eng`

```bash
$ starvault policy write team-eng -<<EOF
path "secret/data/team/eng" {
  capabilities = [ "create", "read", "update", "delete"]
}
EOF
```

2. Создайте внутреннюю группу `engineers` и добавьте сущность `bob-smith` в качестве члена группы и прикрепите `team-eng`.

```bash
$ starvault write identity/group name="engineers" \
     policies="team-eng" \
```

```
      member_entity_ids=$(cat entity_id.txt) \
      metadata=team="Engineering" \
      metadata=region="North America"
```

Пример вывода:

Key	Value
`id`	`0cd76d41-fe36-8c7b-4758-d36bbc212650`
`name`	`engineers`

Теперь, когда вы входите в систему под именем `bob` или `bsmith`, сгенерированный токен наследует политику группового уровня `team-eng`. Чтобы убедиться в этом, можно выполнить аналогичные тесты, показанные в разделе Тестирование сущности. ---

9.2. Вызов API с использованием cURL

1. Создайте полезную нагрузку запроса API, содержащую политику `team-eng`.

```bash
$ tee payload.json <<EOF
{
   "policy": "path \"secret/data/team/eng\" {\n  capabilities = [
\"create\", \"read\", \"update\", \"delete\"]\n}\n
}
EOF
```

2. Создайте новую политику `team-eng`

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request PUT \
   --data @payload-1.json \
   $STARVAULT_ADDR/v1/sys/policies/acl/team-eng
```

3. Создайте полезную нагрузку API-запроса для создания внутренней группы с именем `engineers`.

```bash
$ tee payload-group.json <<EOF
{
  "name": "engineers",
  "policies": ["team-eng"],
  "member_entity_ids": ["$(cat entity_id.txt)"],
  "metadata": {
    "team": "Engineering",
```

```
    "region": "North America"
  }
}
EOF
```

4. Используйте конечную точку `/identity/group`, чтобы создать внутреннюю группу с именем `engineers` и добавить в нее сущность `bob-smith` в качестве члена группы, а также присоединить `team-eng`.

```
BASH
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request PUT \
   --data @payload-group.json \
   $STARVAULT_ADDR/v1/identity/group | jq -r ".data"
```

Пример вывода:

```
BASH
{
  "id": "3439804b-c82e-1977-a306-c9dc2e97f039",
  "name": "engineers"
}
```

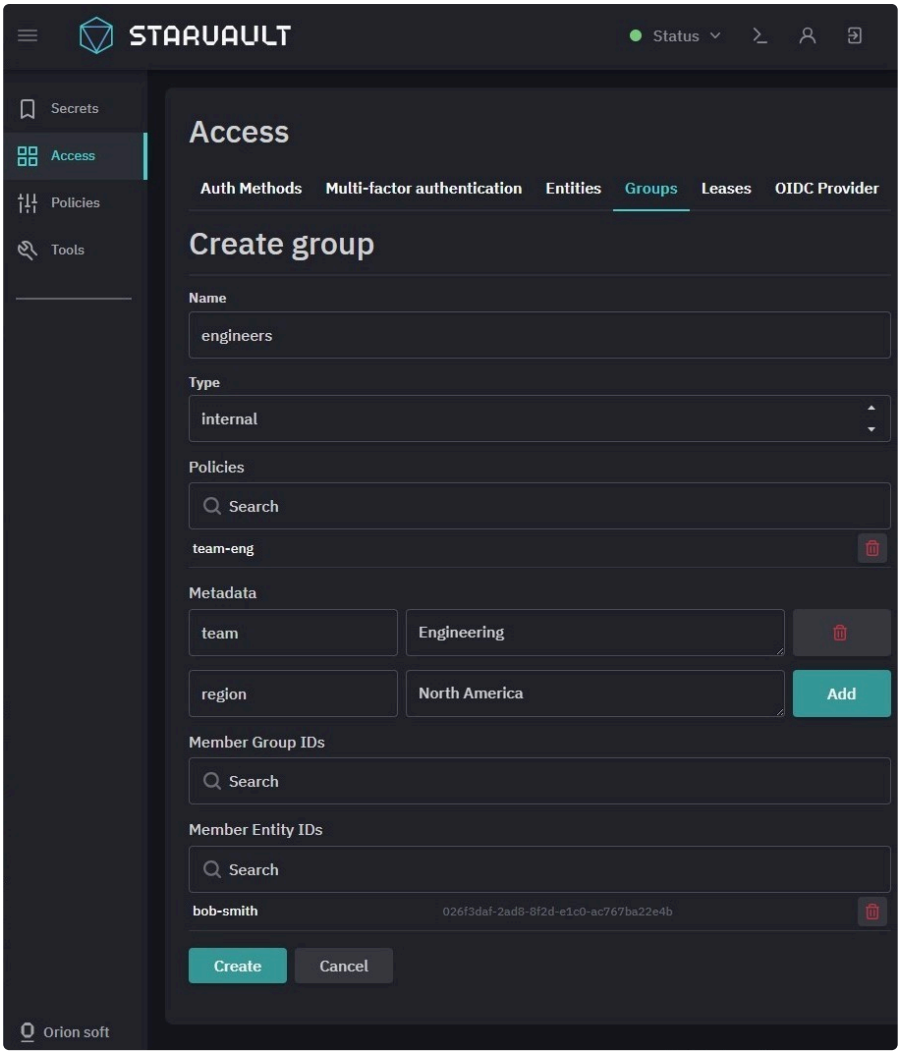
Теперь, когда вы входите в систему под именем `bob` или `bsmith`, сгенерированный токен наследует политику группового уровня `team-eng`. Чтобы убедиться в этом, можно выполнить аналогичные тесты, показанные в разделе Тестирование сущности. ---

9.3. Веб-интерфейс

1. Перейдите на вкладку **Policies**, а затем выберите **Create ACL policy**.
2. Введите `team-eng` в поле **Name** и вставьте следующую политику в текстовое поле **Policy**.

```
path "secret/data/team/eng" {
  capabilities = [ "create", "read", "update", "delete"]
}
```

3. Нажмите **Create Policy** для завершения
4. Нажмите на вкладку **Access** и выберите **Groups**.
5. На левой панели навигации нажмите **Groups* и выберите *Create group**.
6. Добавьте информацию по группе, как показано ниже.



7. Нажмите **Create**.

9.4. Резюме

По умолчанию StarVault создает внутреннюю группу. При создании внутренней группы вы указываете членов группы, а не псевдоним группы. Псевдоним группы является связующим звеном между StarVault и внешними поставщиками идентификационных данных (например, LDAP и т. д.). Поэтому вы определяете псевдонимы групп только при создании внешних групп. Для внутренних групп вы указываете `member_entity_ids` и/или `member_group_ids`.

10. Создание внешней группы

Обычно организации используют такие методы аутентификации, как LDAP для аутентификации пользователей StarVault, а членство отдельных пользователей в группах определяется собственной идентификацией провайдера.

Чтобы управлять авторизацией на уровне группы, можно создать внешнюю группу для связи StarVault с внешним поставщиком идентификационных данных (auth-провайдером) и прикрепить к группе соответствующие политики.

Пример сценария: Любой пользователь, принадлежащий к команде `training` в организации userpass, `example-inc`, имеет право выполнять все операции над путем `secret/education`.

В этом сценарии предполагается, что организация GitHub, example-inc, существует, как и команда `training` в этой организации.

10.1. CLI команда

1. Создайте новую политику `education`.

```
$ starvault policy write education -<<EOF
path "secret/data/education" {
  capabilities = [ "create", "read", "update", "delete", "list" ]
}
EOF
```

2. Включите метод аутентификации GitHub

```
$ starvault auth enable github
```

3. Получите указатель монтирования для метода аутентификации GitHub и сохраните его в файле с именем `accessor_github.txt`.

```
$ starvault auth list -format=json | jq -r '.["github/"].accessor' >
accessor_github.txt
```

4. Настройте указатель на организацию GitHub

Используйте реальное имя организации Github, к которой вы хотите подключиться. В качестве примера ниже используется вымышленное имя организации `example-inc`. Если вы не измените это значение, попытка входа в систему в конце этого раздела не будет успешной.

Пример:

```
$ starvault write auth/github/config organization=example-inc
```

5. Создайте внешнюю группу и сохраните сгенерированный идентификатор группы в файле `group_id.txt`.

Пример:

```
starvault write -format=json identity/group name="education" \
     policies="education" \
     type="external" \
```

```
     metadata=organization="Product Education" | jq -r ".data.id" >
group_id.txt
```

6. Создайте псевдоним группы, где `canonical_id` - это идентификатор группы, а имя должно быть реальным именем существующей команды GitHub.

> Имя должно быть реальным именем команды GitHub, существующей в организации Github, которую вы настроили ранее.

BASH

```
$ starvault write identity/group-alias name="training" \
     mount_accessor=$(cat accessor_github.txt) \
     canonical_id="$(cat group_id.txt)"
```

Пример вывода:

Key	Value
`canonical_id`	`66818a45-ef85-0ff3-6c1e-37faf12ea55e`
`id`	`578944f0-dcfd-29fd-a763-d3f9431512d7`

7. Попробуйте войти в систему как пользователь GitHub, который является частью группы и введите персональный токен доступа пользователя

BASH

```
$ starvault login -format=json -method=github
```

Пример вывода:

BASH

```
{
  "request_id": "fcc10086-9033-dc21-2ddb-d420fbabc474",
  "lease_id": "",
  "lease_duration": 0,
  "renewable": false,
  "data": null,
  "warnings": null,
  "auth":
  {
    "client_token": "s.7QKSeasAME.mD5Ve",
    "accessor": "yUBG78LRdLiZNkBNNTe7OEp1.mD5Ve",
    "policies": [
      "default",
      "education"
    ],
    "token_policies": [
      "default"
    ],
```

```
    "identity_policies": [
      "education"
    ],
    "metadata":
    {
      "org": "example-inc",
      "username": "bobbysmith_developer"
    },
    "orphan": true,
    "entity_id": "58a16f6c-ee92-0cd2-f508-4b2fdb093c07",
    "lease_duration": 3600,
    "renewable": true,
    "mfa_requirement": null
  }
}
```

10.2. Вызов API с помощью cURL

1. Создайте полезную нагрузку запроса API, содержащую строчную политику `education`.

```
$ tee payload-pol.json <<EOF
{
  "policy": "path \"secret/data/education\" {\n   capabilities =
[\"create\", \"read\", \"update\", \"delete\", \"list\"]\n}"
}
EOF
```

2. Создайте политику `education`.

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request PUT \
   --data @payload-pol.json \
   $STARVAULT_ADDR/v1/sys/policies/acl/education
```

3. Включите метод GitHub Auth на GitHub.

```
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data '{"type": "github"}' \
   $STARVAULT_ADDR/v1/sys/auth/github
```

4. Настройте метод аутентификации GitHub, задав организацию.

Используйте реальное имя организации Github, к которой вы хотите подключиться. В качестве примера ниже используется вымышленное имя организации `example-inc`. Если вы не измените это значение, попытка входа в систему в конце этого раздела не будет успешной.

*Пример:

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data '{"organization": "example-inc"}' \
   $STARVAULT_ADDR/v1/auth/github/config
```

5. Получите значение указателя github и сохраните его в файле с именем `accessor_github.txt`.

```bash
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   $STARVAULT_ADDR/v1/sys/auth | jq -r '.data | ."github/".accessor' >
accessor_github.txt
```

6. Создайте полезную нагрузку запроса API для создания внешней группы.

Пример:

```bash
$ tee payload-edu.json <<EOF
{
  "name": "education",
  "policies": ["education"],
  "type": "external",
  "metadata": {
    "organization": "Product Education"
  }
}
EOF
```

7. Создайте внешнюю группу и сохраните сгенерированный идентификатор группы в файле `group_id.txt`.

```bash
curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data @payload-edu.json \
   $STARVAULT_ADDR/v1/identity/group | jq -r ".data.id" > group_id.txt
```

8. Создайте полезную нагрузку API-запроса для создания псевдонима группы для команды, существующей в вашей организации Github.

```bash
$ tee payload-training.json <<EOF
{
  "canonical_id": "$(cat group_id.txt)",
  "mount_accessor": "$(cat accessor_github.txt)",
```

```
  "name": "training"
}
EOF
```

Имя должно быть реальным именем команды GitHub, существующей в организации Github, которую вы настроили ранее.

9. Создайте псевдоним группы.

```
BASH
$ curl --header "X-Vault-Token: $STARVAULT_TOKEN" \
   --request POST \
   --data @payload-training.json \
   $STARVAULT_ADDR/v1/identity/group-alias | jq -r ".data"
```

Пример вывода:

```
BASH
{
   "canonical_id": "81297d74-d67e-afe6-8a57-d76769430796",
   "id": "2891f4e7-554a-2fd7-82a4-3fe890266f75"
}
```

10. Попробуйте войти в систему как пользователь GitHub, входящий в группу, и введите персональный токен доступа пользователя.

```
BASH
curl --request POST \
   --data '{"token": "<actual-personal-access-token>"}' \
   $STARVAULT_ADDR/v1/auth/github/login
```

Пример вывода:

```
BASH
{
  "request_id": "1b3D5f-5b4b-47d5-3bb4-83b2f3cfb4a1",
  "lease_id": "",
  "renewable": false,
  "lease_duration": 0,
  "data": null,
  "wrap_info": null,
  "warnings": null,
  "auth":
  {
    "client_token": "s.IqIsLMNOPwLPjRmUVam.tj0yZ",
    "accessor": "63nn6QRSTU3ASzMoiUP4MkV2.tj0yZ",
    "policies": [
      "default",
      "education"
    ],
    "token_policies": [
      "default"
```

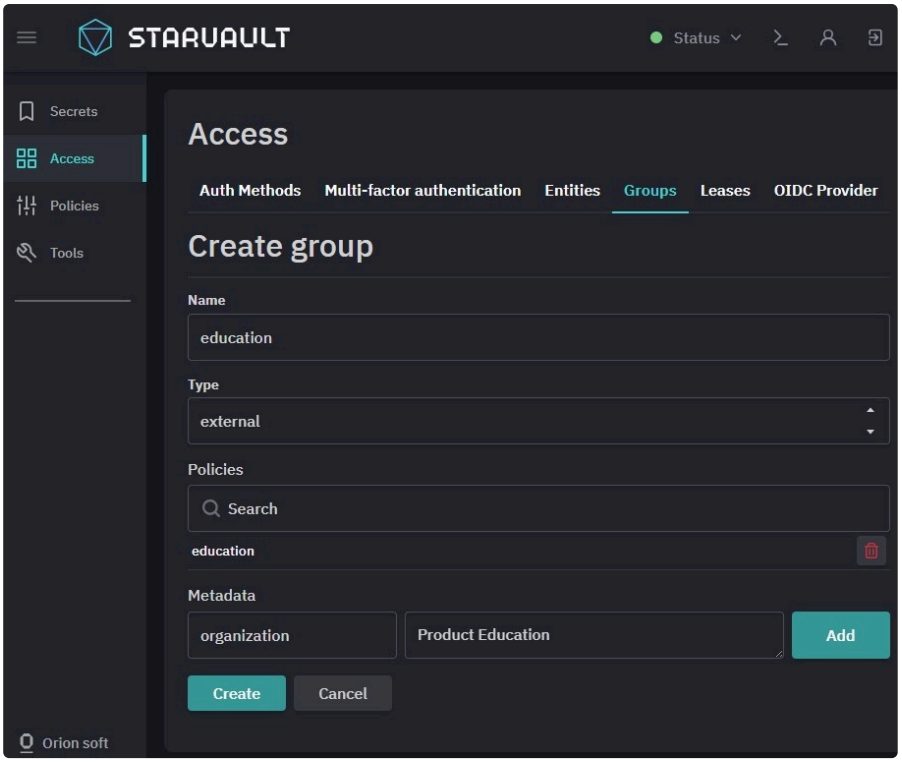
```
    ],
    "identity_policies": [
      "education"
    ],
    "metadata":
    {
      "org": "example-inc",
      "username": "bobbysmith_developer"
    },
    "lease_duration": 3600,
    "renewable": true,
    "entity_id": "f9030f4b-e310-c54b-fc0a-e7f620cd17ac",
    "token_type": "service",
    "orphan": true,
    "mfa_requirement":null,
    "num_uses":0
  }
}
```


10.3. Веб-интерфейс

1. Нажмите на вкладку **Policies** и выберите **Create ACL policy.**
2. Введите `team-eng` в поле **Name** и вставьте следующую политику в поле **Policy**.

```
path "secret/data/education" {
  capabilities = [ "create", "read", "update", "delete", "list" ]
}
```

3. Нажмите **Create Policy** для завершения
4. Зайдите во вкладку **Access** и выберите **Auth Methods**.
5. Выберите **Enable new method**.
6. Нажмите на кнопку GitHub и кликните **Next**.
7. Оставьте путь `github` и нажмите кнопку **Enable Method**.
8. Зайдите на вкладку **Access** и выберите **Groups**
9. Выберите **Create group**. Введите информацию о группе, как показано ниже.



10. Нажмите **Create**.
11. Выберите **Add alias** и введите `training` в поле **Name**. Выберите **github/(github)** в раскрывающемся списке **Auth Backend**.
12. Нажмите **Create**. ---

10.4. Резюме

На этом этапе любой пользователь GitHub, принадлежащий команде `training` в организации ` example-inc`, может пройти аутентификацию в Vault. Сгенерированный токен для пользователя содержит политику `education`.

11. Очистка

Если вы хотите очистить свое окружение после завершения обучения, выполните действия, описанные в этом разделе. --- . Снимите значение переменной окружения `STARVAULT_TOKEN`.

+

```
$ unset STARVAULT_TOKEN
```

BASH

1. Снимите значение переменной окружения `STARVAULT_ADDR`.

```
$ unset STARVAULT_ADDR
```

BASH

2. Вы можете остановить сервер StarVault dev, нажав `Ctrl`+`C`, когда сервер запущен. Или выполните следующую команду.

```
$ pgrep -f starvault | xargs kill
```
BASH

12. Зачем создавать сущности?

В этом руководстве в качестве примера, демонстрирующего использование сущностей и групп, использовался пользователь Боб Смит. Однако эта концепция применима и к приложениям.

Рассмотрим сценарий, в котором приложение использует несколько методов авторизации для входа в StarVault. Если не определена сущность, привязывающая эти несколько идентификаторов авторизации к приложению, сервер StarVault не сможет узнать, что одно и то же приложение вошло в StarVault с помощью разных методов авторизации. В результате каждый вход будет считаться новым клиентом.

Миграция монтируемых точек

В контексте StarVault **миграция монтируемых** точек означает перемещение путей в StarVault, где хранится определенный тип секретных данных (пароли, ключи, сертификаты), управляемых конкретным механизмом хранения. Миграция производится из одного расположения в другое внутри StarVault.

1. Причины миграции точек монтирования

- **Переименование для соответствия стандартам организации:** В StarVault может возникнуть необходимость переименования точек монтирования для приведения их в соответствие с внутренними стандартами организации.

2. Асинхронное поведение

Стоит отметить, что конечная точка `sys/remount` работает асинхронно. При вызове этой конечной точки начинается процесс миграции, и API возвращает идентификатор миграции. Этот идентификатор можно использовать для мониторинга статуса миграции через конечную точку `sys/remount/status`.

3. Аренды

В контексте StarVault термин **аренды** относится к временным разрешениям на доступ к секретам или аутентификационным данным. Это механизм, при котором выдаваемые секреты или токены имеют **ограниченный срок действия** и требуют продления или повторной авторизации.

Операция переноса точки монтирования приведёт к аннулированию всех существующих аренд на этой точке. Если точка монтирования содержит большое количество активных аренд, процесс их аннулирования и сам перенос, может занять значительное время. Все динамические секреты, связанные с точкой монтирования секретов и все токены, относящиеся к точке монтирования аутентификации, будут отозваны в ходе переноса.

4. Конфигурации

Операция переноса сохранит все настройки точки монтирования.

Пусть была создана роль с правами только для чтения на точке монтирования базы данных или пользователь в точке монтирования `userpass`. В данном случае права останутся доступными по новому пути после переноса.

5. Очистка после переноса

После выполнения переноса администраторам StarVault необходимо учитывать два основных аспекта по очистке.

5.1. Политики

Перенос точки монтирования может сделать политики некорректными, если они ссылаются на устаревшие пути.

- Любая политика, ссылающаяся на перемещенный движок секретов, должна быть обновлена с учетом нового пути точки монтирования.
- Если политика больше не должна указывать на исходный путь, необходимо удалить старую ссылку.
- Политики, привязанные к ролям на точке монтирования аутентификации, должны быть проверены на предмет ссылок на недействительные пути.

6. Квоты

Если у точки монтирования были установлены ограничения по скорости запросов или количеству активных аренд, после переноса квоты будут обновлены с учетом нового пути точки монтирования.

7. Фильтры путей

Операция переноса будет учитывать существующие фильтры путей. Рассмотрим случай:

1. Если точка монтирования была специально запрещена в определённом кластере и затем перемещена, она станет доступной в этом кластере.

Настройка политики генерации паролей для секретных механизмов

StarVault поддерживает настраиваемую генерацию паролей, определяемую политикой паролей. Политика содержит правила и требования, которым должен соответствовать пароль, и может предоставлять этот пароль непосредственно через новую конечную точку или через механизмы секретов.

В учебном пособии по динамическим секретам вы настроили StarVault на генерацию динамических учетных данных для базы данных PostgreSQL. В этом руководстве вы узнаете, как настроить StarVault на генерацию паролей, соответствующих стандартам вашей организации.

Если вы не знаете, как настроить StarVault для динамических учетных данных, прочтите руководство по движку секретов базы данных, прежде чем начать.

1. Описание сценария использования

В OrionSoft существует политика паролей, которая должна использоваться для всех приложений. Механизм секретов базы данных StarVault генерирует пароли, которые соответствуют шаблону по умолчанию.

Механизмы секретов с поддержкой политик паролей:

- Active Directory
- LDAP
- RabbitMQ
- Все базы данных

Вы можете изменить стандартный шаблон, чтобы он подходил для таких критериев, как:

- Длина, которая меньше или превышает длину по умолчанию
- Различные наборы символов
- Частота символов
- Требования к расположению символов
- Запрещенные повторы
- Запрещенные слова из словаря

Созадем пользовательскую политику паролей, которая будет соответствовать стандартам безопасности OrionSoft. Требования к паролям следующие:

- Длина 20 символов
- Несколько наборов символов
- Должно быть минимальное количество символов из каждого набора

2. Требования к инфраструктуре

Для реализации сценария, описанного в данном руководстве, вам необходимо:

- Установленный StarVault;
- Развернутая база данных Postgres.

3. Требования к политике

В рамках данного руководства вы можете использовать свой `root` токен для работы с StarVault. Однако рекомендуется использовать `root` токен только для первоначальной настройки или в чрезвычайных ситуациях. В качестве лучшей практики используйте токены с соответствующим набором политик в зависимости от вашей роли в организации.

Для выполнения всех задач, продемонстрированных в этом руководстве, вам нужна политика для оперативной группы, позволяющая настраивать механизм секретов базы данных и управлять политиками паролей.

3.1. Политика администратора StarVault

Данная политика позволяет настраивать механизм секретов базы данных и управлять политиками паролей.

BASH

```
# Mount the database secret engine
path "sys/mounts/database"
{
  capabilities = [ "create", "update", "delete" ]
}

# Manage leases for the database secrets engine readonly role
path "sys/policies/password/example/generate"
{
  capabilities = [ "read" ]
}
```

```
# Generate a password using the example password policy
path "sys/policies/password/example"
{
  capabilities = [ "read", "create", "delete", "update", "list" ]
}

# Configure the database secrets engine and create roles
path "database/*"
{
  capabilities = [ "create", "read", "update", "delete", "list" ]
}
```

Если вы не знакомы с политиками, пройдите учебное пособие по политикам.

4. Создайте лабораторию

1. Разверните контейнер PostgreSQL

```bash
docker run -d \
  --name learn-postgres \
  --env POSTGRES_USER=root \
  --env POSTGRES_PASSWORD=rootpassword \
  --env POSTGRES_DB=postgres \
  --publish 5432:5432 \
  --restart unless-stopped \
  postgres:latest
```

2. Выполните следующие команды для настройки роли.

```bash
docker exec -it learn-postgres psql -U root -d postgres -c "CREATE ROLE ro NOINHERIT;"
docker exec -it learn-postgres psql -U root -d postgres -c "GRANT SELECT ON ALL TABLES IN SCHEMA public TO ro;"
```

3. Разверните контейнер StarVault

```bash
docker run \
  --name learn-starvault \
  --env 'VAULT_DEV_ROOT_TOKEN_ID=root' \
  --env 'VAULT_DEV_LISTEN_ADDRESS=0.0.0.0:8200' \
  --publish 8200:8200 \
  --restart unless-stopped \
  --detach \
  hub.orionsoft.ru/public/starvault:v1.2.0 server -dev
```

4. Экспортируйте переменную окружения для `starvault` CLI, чтобы обратиться к серверу StarVault.

```bash
$ export STARVAULT_ADDR='http://127.0.0.1:8200'
```

5. Экспортируйте переменную окружения для `starvault` CLI, чтобы аутентифицироваться на сервере StarVault.

```bash
$ export STARVAULT_TOKEN=root
```

6. Включите механизм секретов **database**.

```bash
$ starvault secrets enable database
```

7. Настройте механизм секретов **database**.

```bash
$ starvault write database/config/postgres \
    plugin_name=postgresql-database-plugin \
    connection_url="postgresql://{{username}}:
{{password}}@172.17.0.2:5432/postgres?sslmode=disable" \
    allowed_roles="readonly" \
    username="root" \
    password="rootpassword" \
    max_open_connections=4 \
    max_idle_connections=0 \
    max_connection_lifetime="0s"
```

8. Создайте роль для механизма секретов **database**.

```bash
$ starvault write database/roles/readonly \
    db_name=postgres \
    creation_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD
'{{password}}' VALID UNTIL '{{expiration}}' INHERIT;" \
    creation_statements="GRANT ro TO \"{{name}}\";" \
    default_ttl="1h" \
    max_ttl="24h"
```

StarVault и PostgreSQL запущены и настроены. Вы готовы продолжить обучение.

---

5. Запрос учетных данных с политикой паролей по умолчанию

Каждый поддерживаемый механизм секретов имеет политику паролей по умолчанию, которая генерирует пароли.

Сгенерируйте учетные данные для роли `readonly`.

```bash
starvault read database/creds/readonly
```

Пример вывода:

Key	Value
`lease_id`	`database/creds/readonly/5ZJXSIyWN5k4A8u6AMZs5I0s`
`lease_duration`	`1h`
`lease_renewable`	`true`
`password`	`-LL23iRQqsAD0YmS1eq3`
`username`	`v-token-readonly-xboXz3doC1jBe9ntlKaA-1737490496`

В учетных данных отображаются созданные `username` и `password`. Созданный пароль соответствует политике паролей по умолчанию для механизма секретов.

6. Определение политики паролей

1. Создайте файл политики с именем `example_policy.hcl`.

```bash
$ tee example_policy.hcl <<EOF
length=20

rule "charset" {
  charset = "abcdefghijklmnopqrstuvwxyz"
  min-chars = 1
}

rule "charset" {
  charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  min-chars = 1
}

rule "charset" {
  charset = "0123456789"
  min-chars = 1
}

rule "charset" {
  charset = "!@#$%^&*"
```

```
  min-chars = 1
}
EOF
```

Политики используют язык конфигурации HCL. Поле `length` устанавливает длину возвращаемого пароля в 20 символов. Каждая строфа правил определяет набор символов и минимальное количество повторений этих символов в генерируемом пароле. Эти правила являются совокупными, поэтому каждое из них добавляет больше требований к генерируемому паролю.

2. Создайте политику паролей StarVault с именем `example` и правилами политики паролей, определенными в файле `example_policy.hcl`.

```
$ starvault write sys/policies/password/example policy=@example_policy.hcl
Success! Data written to sys/polices/password/example
```

Эта политика доступна для генерации пароля напрямую или с помощью ссылки в `example` при настройке поддерживаемых механизмов секретов.

3. Сгенерируйте пароль из `example` политики паролей.

```
$ starvault read sys/policies/password/example/generate
```

Key	Value
`password`	`#v!RQDHxHunJ1TUmCyys`

Созданный пароль соответствует требованиям:

- длина 20 символов
- не менее 1 прописного символа
- не менее 1 строчного символа
- не менее 1 цифры
- не менее 1 символа

7. Настройка пользовательской политики паролей

1. Выведите текущую конфигурацию механизма секретов базы данных

```
$ starvault read database/config/postgres
```

Key	Value
allowed_roles	[readonly]
connection_details	map[connection_url:postgresql://{{username}}:{{password}}@172.17.0.2:5432/postgres?sslmode=disable max_connection_lifetime:0s max_idle_connections:0 max_open_connections:4 username:root]
password_policy	n/a
plugin_name	postgresql-database-plugin
plugin_version	n/a
root_credentials_rotate_statements	[]
verify_connection	true

Первоначально настроенный механизм секретов не содержит политики паролей.

2. Настройте механизм секретов на использование политики паролей `example`.

```
$ starvault write database/config/postgres password_policy="example"
Success! Data written to: database/config/postgres
```

3. Просмотрите обновленную конфигурацию механизма секретов базы данных.

```
$ starvault read database/config/postgres
```

Пример вывода:

Key	Value
allowed_roles	[readonly]
connection_details	map[connection_url:postgresql://{{username}}:{{password}}@172.17.0.2:5432/postgres?sslmode=disable max_connection_lifetime:0s max_idle_connections:0 max_open_connections:4 username:root]
password_policy	example

Key	Value
…snip…	

Сейчас механизмы секретов генерируют пароли, соответствующие политике паролей `example`.

4. Сгенерируйте учетные данные для роли `readonly` с помощью политики паролей `example`.

```
$ starvault read database/creds/readonly
```

Key	Value
---	-----
`lease_id `	`database/creds/readonly/AppmaMdS5mtMPwaZOnNnX209`
`lease_duration`	`1h`
`lease_renewable`	`true`
`password`	`ba8oWN@106l6uffb*GbY`
`username`	`v-token-readonly-W1guRN3Sr2SEw4Oj0uJU-1737491327`

В учетных данных отображаются созданные `username` и `password`. Созданный `password` соответствует `example` политики паролей, определенному в конфигурации механизма секретов.

8. Очистка

1. Остановите контейнеры.

```
$ docker stop learn-postgres learn-starvault
```

2. Снимите настройки переменных окружения.

```
$ unset STARVAULT_ADDR STARVAULT_TOKEN
```

3. Удалите файл политики паролей `example_policy.hcl`.

```
$ rm example_policy.hcl
```

9. Резюме

Вы запросили учетные данные у механизма секретов базы данных, который сгенерировал учетные данные с политикой паролей по умолчанию. Затем вы определили политику паролей и сгенерировали пароль. Наконец, вы обновили конфигурацию механизма секретов, чтобы использовать пользовательскую политику паролей.