



Assess Vault Tokens



Objective 3 – Assess Vault Tokens

Objective 3a – Describe Vault Token

Objective 3b – Differentiate between service and batch tokens. Choose one based on use-case

Objective 3c – Describe root token uses and lifecycle

Objective 3d – Define token accessors

Objective 3e – Explain time-to-live

Objective 3f – Explain orphaned tokens

Objective 3g – Create tokens based on need



Vault Interfaces

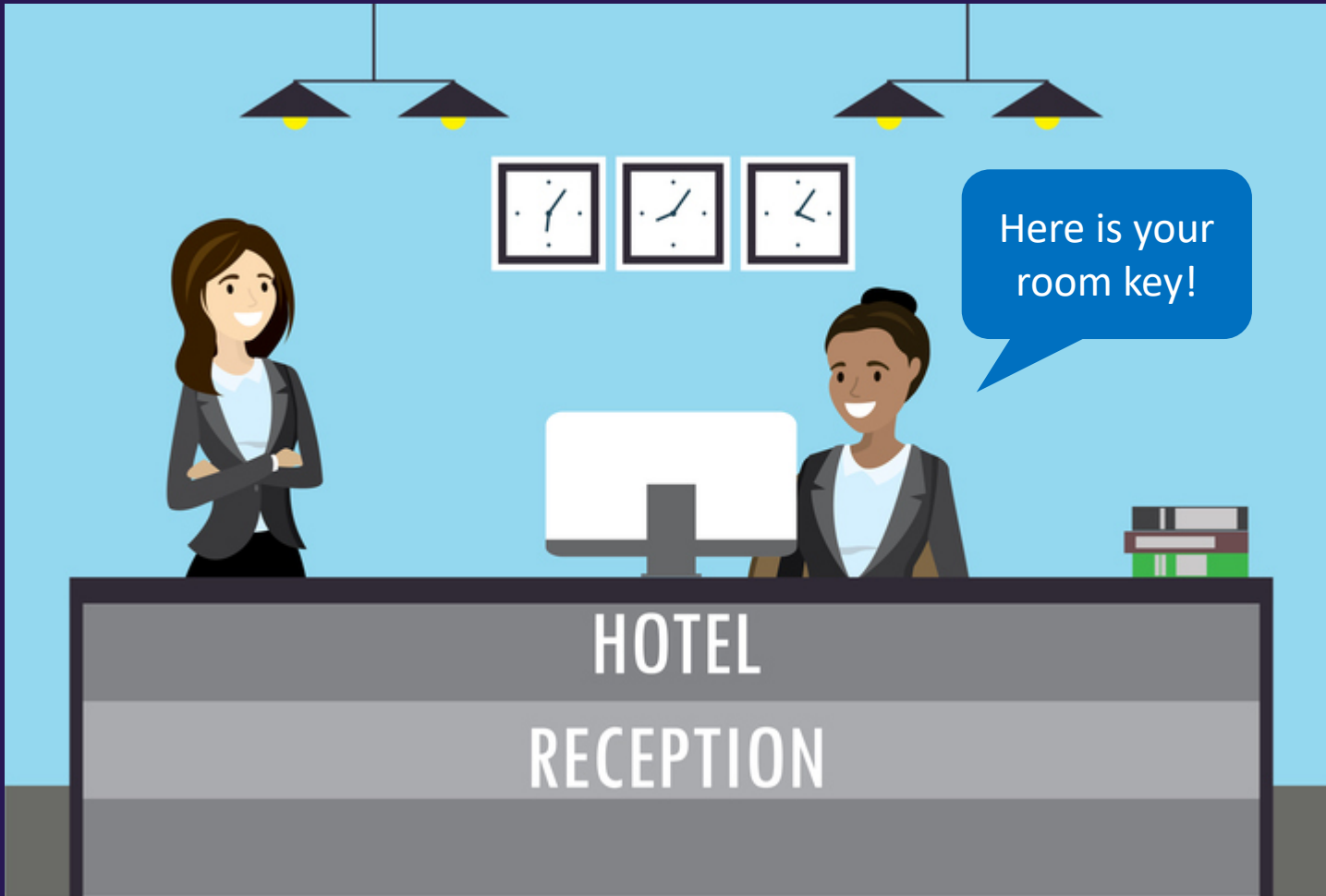
Authentication



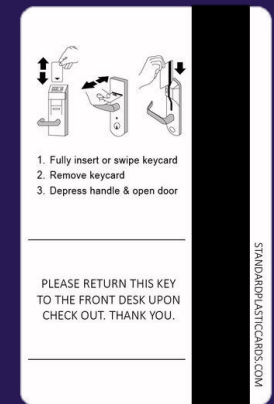
Vault Interfaces



VALID FOR 3 DAYS



Vault Interfaces

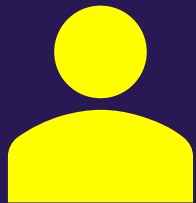
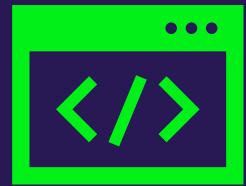


We present our key.
We don't authenticate again



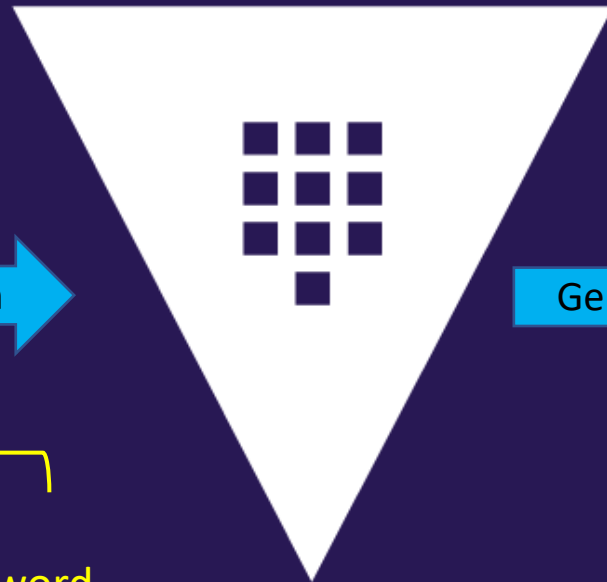
Vault Interfaces

Token Generation



Authentication

Username & Password
RoleID & Secret ID
TLS Certificate
Integrated Cloud Creds



Generate Token

1010
1010

Vault Path(s)
Read/Write/Delete/List



VALID FOR 4 HOURS
(TTL)



Vault Interfaces

Token Usage



We present our token.
We don't authenticate again



Vault Tokens

- Tokens are the **core method for authentication**
 - Most operations in Vault require an existing token (not all, though)
 - Accessing a login path doesn't, for example
- The **token auth method** is responsible for creating and storing tokens
 - The token auth method **cannot** be disabled
 - Tokens can be used directly, or they can be used with another auth method
 - Authenticating with an external identity (e.g. LDAP) dynamically generate tokens
- Tokens have one or more policies attached to control what the token is allowed to perform



Types of Tokens

- **service** tokens are the default token type in Vault
 - They are **persisted to storage** (heavy storage reads/writes) ✓
 - Can be **renewed**, **revoked**, and **create child tokens**
 - Most often, you'll be working with service tokens
- **batch** tokens are encrypted binary large objects (blobs)
 - Designed to be **lightweight** & **scalable**
 - They are **NOT** persisted to storage but they are not fully-featured ✗
 - Ideal for **high-volume operations**, such as encryption
 - Can be used for DR Replication **cluster promotion** as well



Comparing Token Types

Characteristic	Service Tokens	Batch Tokens
Can be Root Tokens	✓	✗
Can Create Child Tokens	✓	✗
Renewable	✓	✗
Can be Periodic	✓	✗
Can have Explicit Max TTL	✓	✗
Has an Accessor	✓	✗
Has Cubbyhole	✓	✗
Revoked with Parent (if not orphan)	✓	Stops Working
Dynamic Secrets Lease Assignment	Self	Parent (if not orphan)
Can be used across Performance Replication clusters	✗	✓
Creation scales with Performance Standby Node Count	✗	✓
Performance Cost	Heavyweight Multiple Writes Per Token Creation	Lightweight No Storage Cost for Token Creation

Primary
Differences



Information/Metadata Attached to a Token

Tokens carry information and metadata that determines how the token can be used, what type of token, when it expires, etc.

- Accessor
- Policies
- TTL
- Max TTL
- Number of Uses Left
- Orphaned Token
- Renewal Status



Information/Metadata Attached to a Token

```
TERMINAL
$ vault token lookup s.d1BCdhug8buTgAnSZhtPm8Hp

Key          Value
---          -
accessor     5mXJQjjQvG44ymJZ0lSHihTG
creation_time 1630436317
creation_ttl  768h
display_name token
entity_id    n/a
expire_time   2021-10-02T14:58:37.2194177-04:00
explicit_max_ttl 0s
id           s.d1BCdhug8buTgAnSZhtPm8Hp
issue_time   2021-08-31T14:58:37.2194177-04:00
meta         <nil>
num_uses     0
orphan       false
path         auth/token/create
policies     [default user]
renewable    true
ttl          767h59m47s
type         service
```



Token Hierarchy

- Each token has a **time-to-live (TTL)**
 - Exception: root token has no TTL
- Tokens are **revoked** once reached its TTL unless renewed
 - Once a token reaches its **max TTL**, it gets revoked
 - May be revoked early by manually revoking the token
 - When a parent token is revoked, all of its children are revoked as well



Token Hierarchy



Controlling Token Lifecycle



What if I don't like the default behavior of the token hierarchy or my app can't handle it?

There are other options



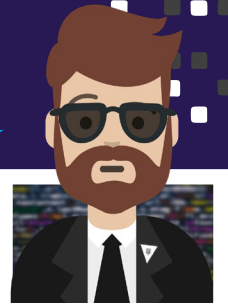
Controlling Token Lifecycle



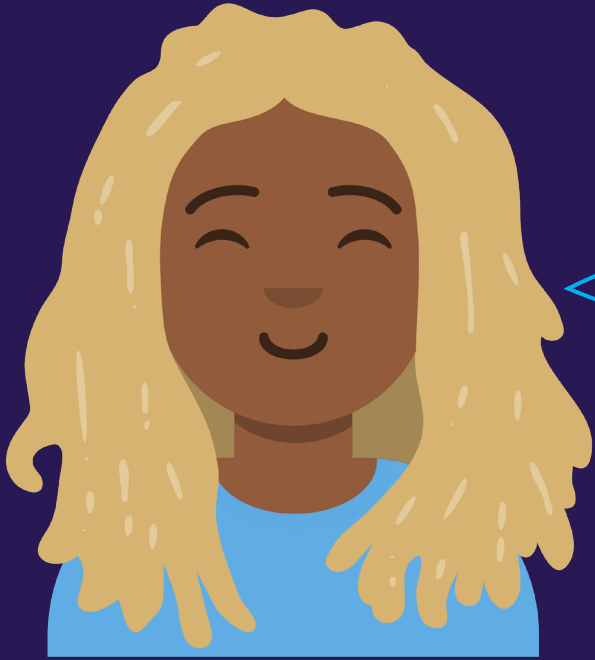
App Developer

I have a long-running app which cannot handle the regeneration of a token or secret

Use a Periodic Service Token



Controlling Token Lifecycle



Principal Engineer

I need a token that gets revoked automatically after one use

Use a Service Token with a Use Limit



Controlling Token Lifecycle



DevOps Engineer

My app can't use a token where its expiration is influenced by its parent

You can use an Orphan Token



Controlling Token Lifecycle

Summary

Challenge	Solution
I have a long-running app which cannot handle the regeneration of a token or secret	Periodic Service Token
I need a token that gets revoked automatically after one use	Service Token with Use Limit
My app can't use a token where its expiration is influenced by its parent	Orphan Service Token



Periodic Token

- When having a token be revoked would be problematic:
 - **Root** or **sudo** users have the ability to generate periodic tokens

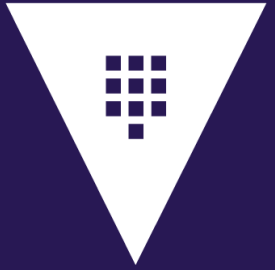
```
policy.hcl
path "auth/token/create" {
  capabilities = [ "create", "read", "update", "delete", "sudo" ]
}
```

- Periodic tokens have a TTL, but no max TTL
- Periodic tokens may live for an infinite amount of time, so long as they are renewed within their TTL

This is useful for long-running services/applications that cannot handle regenerating a token



Periodic Token



```
admin-policy.hcl

$ vault token create -policy=training -period=24h

Key          Value
---          -
token        s.2kjqZ12ofDr3efPdtMJ1z5dZ
token_accessor 73rjN1kmnzwT71pMw9H7p6P9
token_duration 24h
token_renewable true
token_policies ["default" "training"]
identity_policies []
policies       ["default" "training"]
```



No Max TTL



Periodic Token



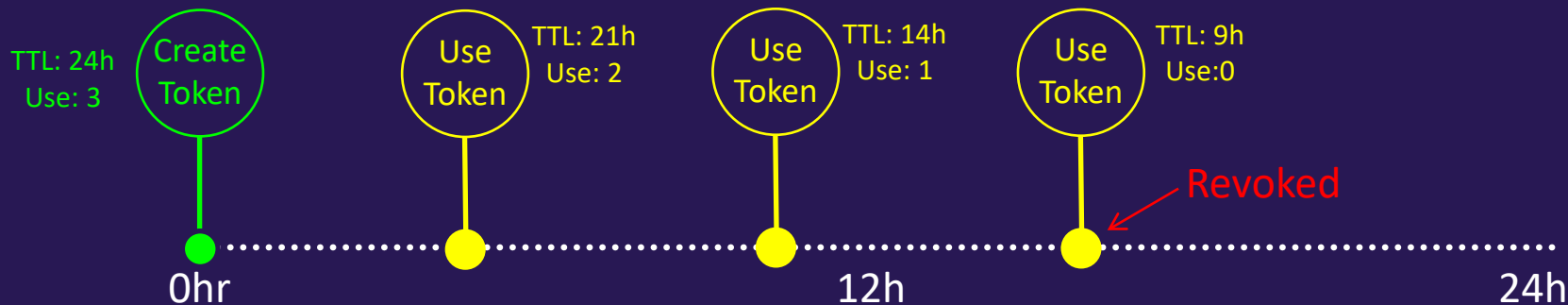
```
admin-policy.hcl

$ vault token lookup s.ya32UrL0ASLkvIOLsM5mV19c
Key          Value
---          -
accessor     73rjN1kmnzwT71pMw9H7p6P9
creation_time 1632751059
creation_ttl  24h
display_name  token
entity_id     n/a
expire_time   2021-09-28T09:57:39.7239753-04:00
explicit_max_ttl 0s
id            s.2kjqZ12ofDr3efPdtMJ1z5dZ
issue_time    2021-09-27T09:57:39.7239753-04:00
meta          <nil>
num_uses      0
orphan        false
path          auth/token/create
period        24h
policies      [default training]
renewable     true
ttl           23h59m28s
type          service
```

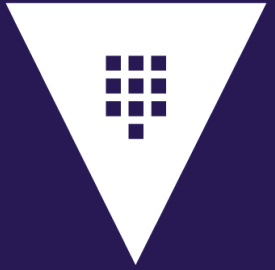


Service Token with Use Limits

- When you want to limit the number of requests coming to Vault from a particular token:
 - Limit the token's number of uses in addition to TTL and Max TTL
 - Use limit tokens expire at the end of their last use, regardless of their remaining TTLs
 - Use limit tokens expire at the end of their TTLs, regardless of remaining uses



Service Token with Use Limits



```
admin-policy.hcl

$ vault token create -policy="training" -use-limit=2
Key          Value
---          -
token       s.516L09Ssk1CQzvKo8ny1G0eu
...

$ vault token lookup s.516L09Ssk1CQzvKo8ny1G0eu
Key          Value
---          -
...
id           s.516L09Ssk1CQzvKo8ny1G0eu
issue_time   2021-12-25T18:35:08.004652-08:00
meta         <nil>
num_uses     2
```



Orphan Token

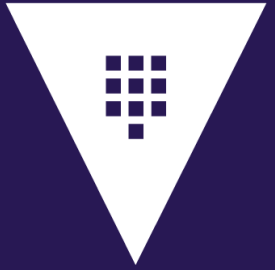
- When the token hierarchy behavior is not desirable:
 - Root or sudo users have the ability to generate orphan tokens

```
policy.hcl
path "auth/token/create-orphan" {
  capabilities = [ "create", "read", "update", "delete", "sudo" ]
}
```

- Orphan tokens are not children of their parent; therefore, do not expire when their parent does
- Orphan tokens still expire when their own Max TTL is reached



Orphan Token



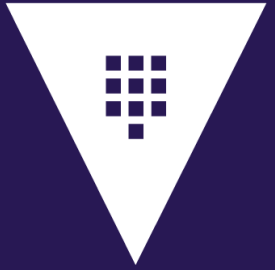
```
admin-policy.hcl

$ vault token create -policy="training" -orphan
Key          Value
---          -
token       s.3rPJCQbGWD9O6uybtTuojjFs
...

$ vault token lookup s.3rPJCQbGWD9O6uybtTuojjFs
Key          Value
---          -
...
id           s.3rPJCQbGWD9O6uybtTuojjFs
issue_time   2018-12-13T18:35:41.02532-08:00
meta         <nil>
num_uses     0
orphan       true
...
```



Setting the Token Type



```
admin-policy.hcl

$ vault token create -policy="training" -period="24h"

Key          Value
---          -
token        s.2kjqZ12ofDr3efPdtMJ1z5dZ
token_accessor 73rjN1kmnzwT71pMw9H7p6P9
token_duration 24h
token_renewable true
token_policies ["default" "training"]
identity_policies []
policies       ["default" "training"]
```



Setting the Token Type

- To configure the AppRole auth method to generate batch tokens:

```
TERMINAL
$ vault auth enable approle
$ vault write auth/approle/role/training policies="training" \
  token_type="batch" \
  token_ttl="60s"
```

- To configure the AppRole auth method to generate periodic tokens:

```
TERMINAL
$ vault write auth/approle/role/jenkins policies="jenkins" \
  period="72h"
```



Managing Tokens in Vault

Command Line Interface (CLI)

Use the `vault token` command

- `capabilities`
- `create`
- `lookup`
- `renew`
- `revoke`

Terminal

```
$ vault token create -ttl=5m -policy=training
```

Key	Value
---	----
token	s.12VNpg40A9tTdCd4V6ODuDRK
token_accessor	1MIaZ4Tn1t57wKXdsfNv7v1m
token_duration	5m
token_renewable	true
token_policies	["default" "training"]
identity_policies	[]
policies	["default" "training"]

Terminal

```
vault token revoke s.tvIb1APJV2BQby01PEw4EgIN  
Success! Revoked token (if it existed)
```



Managing Tokens in Vault

Command Line Interface (CLI)

```
vault token create -policy=training -ttl=24h
```

Type of Vault
object you want
to work with

Subcommand

Define the policy or policies you
want to attach to the token

Define the length
of validity for the
token



Managing Tokens in Vault

Command Line Interface (CLI)

```
vault token create \  
-display_name=jenkins \  
-policy=training,certs \  
-ttl=24h \  
-explicit-max-ttl = 72h
```



Vault command to create token



Give it a friendly name



Assign multiple policies



Specify the TTL



Specify the Maximum TTL



Managing Tokens in Vault

Command Line Interface (CLI)

Create a Token

Terminal

```
$ vault token create -ttl=5m -policy=training

Key                Value
---                -
token              s.12VNpg4OA9tTdCd4V6ODuDRK
token_accessor     1MIaZ4Tn1t57wKXdsfNv7v1m
token_duration     5m
token_renewable    true
token_policies     ["default" "training"]
identity_policies  []
policies           ["default" "training"]
```

Revoke a Token

Terminal

```
$ vault token revoke s.12VNpg4OA9tTdCd4V6ODuDRK
Success! Revoked token (if it existed)
```

Lookup Information About a Token

Terminal

```
$ vault token lookup s.12VNpg4OA9tTdCd4V6ODuDRK

Key                Value
---                -
accessor          1MIaZ4Tn1t57wKXdsfNv7v1m
creation_time     1630613718
creation_ttl      5m
display_name      token
entity_id         n/a
expire_time       2021-09-02T16:23:02.6427677-04:00
explicit_max_ttl  0s
id                s.12VNpg4OA9tTdCd4V6ODuDRK
issue_time        2021-09-02T16:15:18.5177235-04:00
last_renewal      2021-09-02T16:18:02.6427677-04:00
last_renewal_time 1630613882
meta              <nil>
num_uses          0
orphan            false
path              auth/token/create
policies          [default training]
renewable         true
ttl               3m12s
type              service
```



Managing Tokens in Vault

Command Line Interface (CLI)

Look up the capabilities of a token on a particular path

Terminal

```
$ vault token capabilities s.dhtIk8VsE3Mj61PuGP3ZfFrg kv/data/apps/webapp
create, list, read, sudo, update
```

Lookup a Token

Terminal

```
$ vault token lookup s.dhtIk8VsE3Mj61PuGP3ZfFrg
Key          Value
---          -
accessor     INk5tw0tl3N2xs0XZZfPc9Tq
creation_time 1630614230
creation_ttl  5m
policies     [default training]
renewable    true
ttl          3m31s
type         service)
...
```

Abbreviated Output

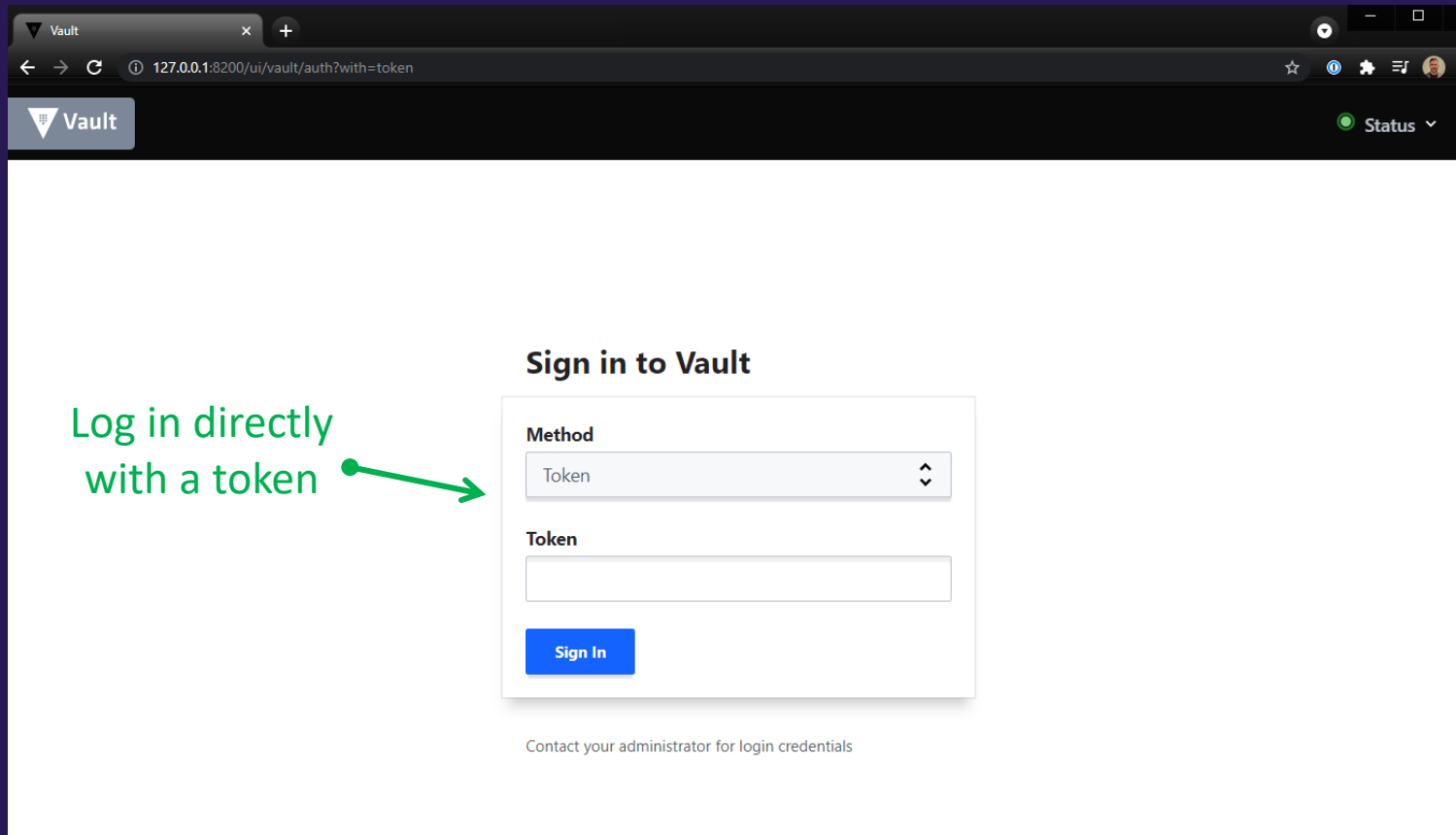
Renew the Token

Terminal

```
$ vault token renew s.dhtIk8VsE3Mj61PuGP3ZfFrg
Key          Value
---          -
token        s.dhtIk8VsE3Mj61PuGP3ZfFrg
token_accessor INk5tw0tl3N2xs0XZZfPc9Tq
token_duration 5m
token_renewable true
token_policies ["default" "training"]
identity_policies []
policies      ["default" "training"]
```

Managing Tokens in Vault

User Interface (UI)



The screenshot shows a web browser window with the URL `127.0.0.1:8200/ui/vault/auth?with=token`. The page title is "Vault" and the status is "Status". The main content area is titled "Sign in to Vault" and contains a form with the following fields:

- Method:** A dropdown menu with "Token" selected.
- Token:** An empty text input field.
- Sign In:** A blue button.

Below the form, there is a link: "Contact your administrator for login credentials".

A green arrow points from the text "Log in directly with a token" to the "Token" dropdown menu.

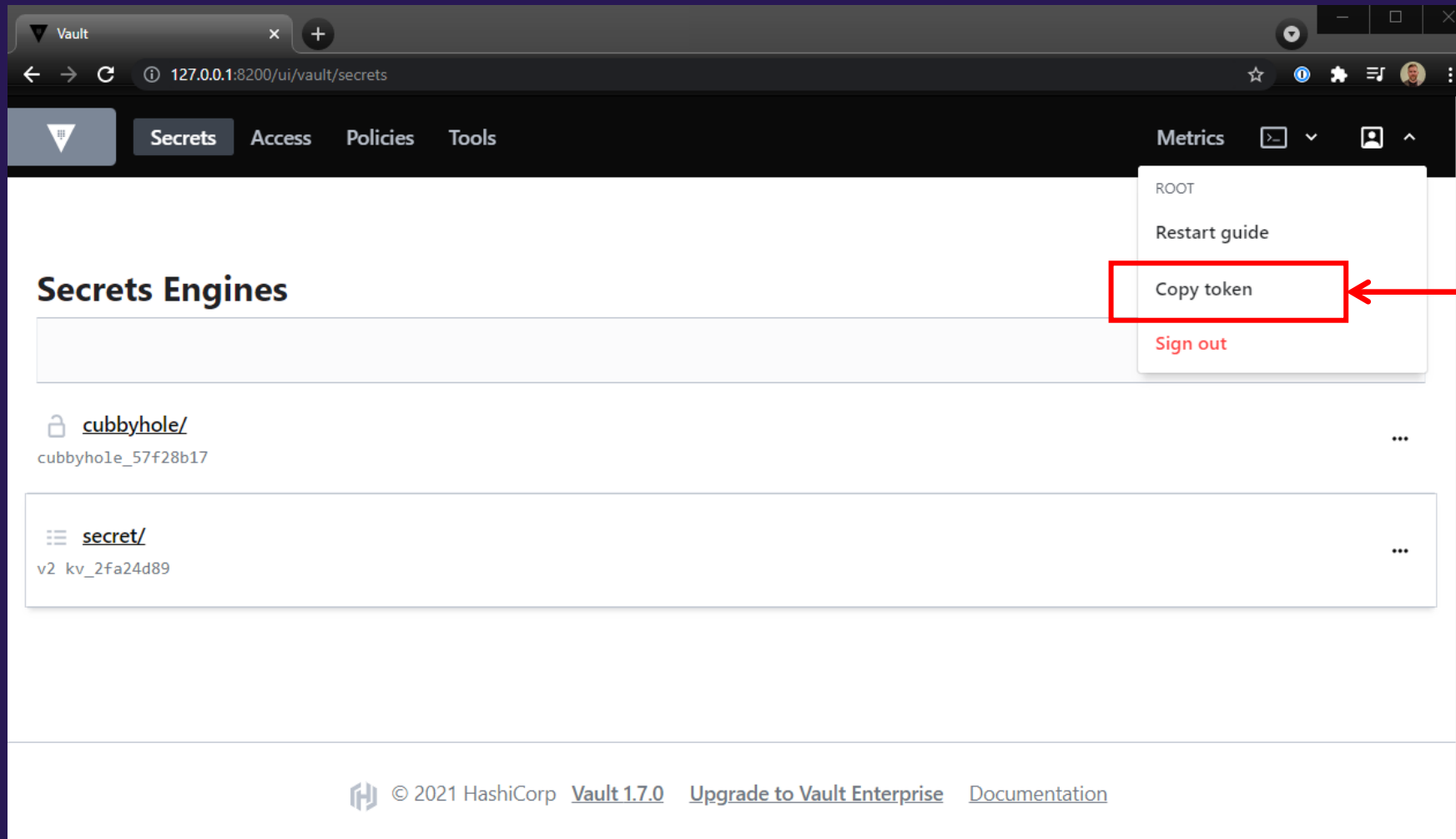
Log in directly
with a token



Managing Tokens in Vault

User Interface (UI)

Copy the Token
You're Using



The screenshot shows the Vault web interface in a browser window. The address bar displays '127.0.0.1:8200/ui/vault/secrets'. The navigation menu includes 'Secrets', 'Access', 'Policies', and 'Tools'. A 'Metrics' dropdown menu is open, showing options: 'ROOT', 'Restart guide', 'Copy token', and 'Sign out'. The 'Copy token' option is highlighted with a red rectangular box. A red arrow points from the text 'Copy the Token You're Using' to this box. Below the menu, the 'Secrets Engines' section is visible, listing two engines: 'cubbyhole/' with token 'cubbyhole_57f28b17' and 'secret/' with token 'v2 kv_2fa24d89'. The footer contains the HashiCorp logo, copyright information for 2021, and links for 'Vault 1.7.0', 'Upgrade to Vault Enterprise', and 'Documentation'.



Managing Tokens in Vault

HTTP API

- Authenticating to Vault with an auth method will result in a response that includes a token
 - Response is in **JSON**, so you can parse the response to get the token
 - Use jq to parse → **.auth.client_token** is the value you want
- Future requests to Vault, such as requests for a secret or to make a configuration change should include this token
 - Token is sent via **X-Vault-Token** header
 - **Authorization Bearer** is also a valid header



Managing Tokens in Vault

HTTP API

TERMINAL

```
$ curl --request POST --data @payload.json http://127.0.0.1:8200/v1/auth/userpass/login/bryan | jq
{
  "request_id": "0b4181fe-0dec-2261-5231-bb3f033387e5",
  "lease_id": "",
  "renewable": false,
  "auth": {
    "client_token": "s.WNS4zL4c4wQJet9KS9KitkHW",
    "accessor": "zsap13bBoQGzB5xVPZFEu3Th",
    "policies": [
      "default",
      "training"
    ],
    "token_policies": [
      "default",
      "training"
    ],
    "metadata": {
      "username": "bryan"
    },
    "lease_duration": 2764800,
    "renewable": true,
    "entity_id": "88669d54-b405-c27a-d468-410a1185eb0d",
    "token_type": "service",
    "orphan": true
  }
}
```



Managing Tokens in Vault

HTTP API

Authenticate and store the resulting token in a file

Terminal

```
$ curl --request POST --data @payload.json http://127.0.0.1:8200/v1/auth/userpass/login/bryan |  
jq -r ".auth.client_token" > token.txt
```

```
$ cat token.txt  
s.dhtIk8VsE3Mj61PuGP3ZfFrg
```

Authenticate and store the resulting token in an environment variable

Terminal

```
$ OUTPUT=$(curl --request POST --data @payload.json  
http://127.0.0.1:8200/v1/auth/userpass/login/bryan)  
  
$ VAULT_TOKEN=$(echo $OUTPUT | jq '.auth.client_token' -j)  
  
$ echo $VAULT_TOKEN  
s.dhtIk8VsE3Mj61PuGP3ZfFrg
```



Managing Tokens in Vault

HTTP API

Client token must be sent in the X-Vault-Token HTTP header

Terminal

```
$ curl --header "X-Vault-Token: s.dhtIk8VsE3Mj61PuGP3ZfFrg" \  
--request POST \  
--data '{ "apikey": "3230sc$832d" }' \  
https://vault.example.com:8200/v1/secret/apikey/splunk
```

Terminal

```
$ curl --header "X-Vault-Token: s.dhtIk8VsE3Mj61PuGP3ZfFrg" \  
--request GET \  
https://vault.example.com:8200/v1/secret/data/apikey/splunk
```



Root Tokens

Root token is a **superuser** that has unlimited access to Vault

- It does NOT have a TTL – meaning it **does not expire**
- Attached to the root policy
- **Note:** Root tokens can create other root tokens that DO have a TTL

Root tokens should NOT be used on a day-to-day basis

- In fact, rarely should a root token even exist
- Once you have used the root token, **it should be revoked**



Root Tokens



Where Do Root
Tokens Come From?



Where do Root Tokens Come From?

vault operator init

Initial root token comes from Vault **initialization**

- Only method of authentication when first deploying Vault
- Used for initial configuration – such as **auth methods** or **audit devices**
- Once your new auth method is configured and tested, **the root token should be revoked**

Terminal

```
$ vault token revoke s.dhtIk8VsE3Mj61PuGP3ZfFrg  
Success! Revoked token (if it existed)
```



Where do Root Tokens Come From?

Existing Root Token

Create a root token from an existing root token

- You can authenticate with a root token and run a `vault token create`

Terminal

```
$ vault login s.lmmOCfNH1HZvvBwxnLErWrhK  
Success! You are now authenticated. The token information displayed below  
is already stored in the token helper. You do NOT need to run "vault login"  
again. Future Vault requests will automatically use this token.
```

Key	Value
token	s.lmmOCfNH1HZvvBwxnLErWrhK
token_accessor	5UNwzGSr1TOGymhERwZeAMgr
token_duration	∞
token_renewable	false
token_policies	["root"]
identity_policies	[]
policies	["root"]

2. Create a new root token

```
$ vault token create
```

Key	Value
token	s.tiRn8HflpBJNssFaSWTTCOI2
token_accessor	anZIDSIUzPUcs6hKKsOdwoXj
token_duration	∞
token_renewable	false
token_policies	["root"]
identity_policies	[]
policies	["root"]

New root token

1. Log in with root token



Where do Root Tokens Come From?

Using Keys

Create a root token using **unseal/recovery keys**

- Helpful if you need to generate a root token in an emergency or a root token is needed for a particular task
- A quorum of unseal key holders can generate a new root token
 - Enforces the "no single person has complete access to Vault"

Step 1

Initialize the root generation

Step 2

Each key holder runs 'generate root' with their unseal key

Step 3

Decode the generated root token



Generating a Root Token

Using Keys

To perform the task, use the **vault operator generate-root** command

Command Options	Description
-generate-otp	Generate and print high-entropy one-time-password
-init	Start a root token generation
-decode=<string>	Decode and output the generated root token
-otp=<string>	OTP code to use with -decode or -init
-status	Print the status of the current attempt
-cancel	Cancel the current attempt



Generating a Root Token

Using Keys – Step 1

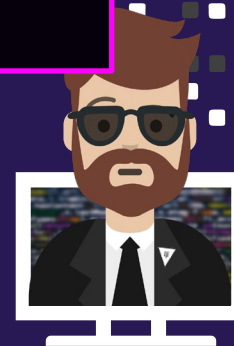
Terminal

```
$ vault operator generate-root -init
```

A One-Time-Password has been generated for you and is shown in the OTP field. You will need this value to decode the resulting root token, so keep it safe.

```
Nonce          5b6e3831-2a45-4695-7757-5810074d36c8
Started        true
Progress       0/1
Complete       false
OTP            E87jF6ZeJo8NjJwvyt17mvKLEr
OTP Length     26
```

One-Time-Password (OTP) gets generated



Generating a Root Token

Using Keys – Step 2

Terminal

```
$ vault operator generate-root
Root generation operation nonce: f8579a51-5138-c31...
Unseal Key (will be hidden):
Nonce      f8579a51-5138-c319-445d-2d3640119f87
Started    true
Progress   1/3
Complete   false
```



Key holders each provide their key until you meet the threshold



Generating a Root Token

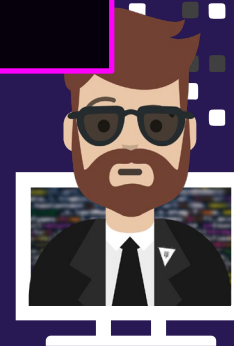
Using Keys – Step 3

Terminal

```
$ vault operator generate-root
Root generation operation nonce: f8579a51-5138-c319...
Unseal Key (will be hidden):
Nonce          f8579a51-5138-c319-445d-2d3640119f87
Started        true
Progress       3/3
Complete       true
Encoded Token   G2NeKUZgXTsYYxILAC9ZFBguPw9ZXBovFAs
```



Encrypted Root Token



Generating a Root Token

Using Keys – Step 4

Terminal

```
$ vault operator generate-root \  
  -otp="hM9q24nNiZfnYIiNvhnGo4UFc3" \  
  -decode="G2NeKUZgXTsYYxILAC9ZFBguPw9ZXBovFAs"
```

```
Root token: s.gXtT3uq9teYf0ZnFQH6hOiw8
```



We Got A Root Token!!!



Token Accessors

Every token has a token accessor that is used as a **reference** to the token

Token accessors can be used to perform **limited actions**

- Look up token properties
- Look up the capabilities of a token
- Renew the token
- Revoke the token

Token accessors cannot be used for authentication to Vault or to perform additional requests



Token Accessors

Root Token Accessor Example

Terminal

```
$ vault login s.cbC7GJ6U6WJaDuDSgkyVcKDv
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.
```

Key	Value
---	-----
token	s.cbC7GJ6U6WJaDuDSgkyVcKDv
token_accessor	K6pHtVc9LbXQdUavg2J1Ixa2
token_duration	∞
token_renewable	false
token_policies	["root"]
identity_policies	[]
policies	["root"]



Token Accessors

Regular Token Accessor Example

Terminal

```
$ vault token create -policy=training -ttl=30m
```

Key	Value
---	-----
token	s.5YmCHHV80mN3dJpzOwvVAYk8
token_accessor	2ogWa36gDH5wsO8VbuxroByx
token_duration	30m
token_renewable	true
token_policies	["default" "training"]
identity_policies	["default" "training"]
token_policies	["default" "training"]



Token Accessors

Viewing the Properties of a Token Using its Accessor

Terminal

```
$ vault token lookup -accessor gFq2UwnJ0jo87kESKwUcl1Ub
```

Key	Value
---	-----
accessor	gFq2UwnJ0jo87kESKwUcl1Ub
creation_time	1632576647
creation_ttl	30m
display_name	token
entity_id	n/a
expire_time	2021-09-25T10:00:47.0615482-04:00
explicit_max_ttl	0s
id	n/a
issue_time	2021-09-25T09:30:47.0615482-04:00
meta	<nil>
num_uses	0
orphan	false
path	auth/token/create
policies	[default training]
renewable	true
ttl	29m18s
type	service



Token Accessors

Revoking a Token using the Accessor

Terminal

```
$ vault token create -policy=training -ttl=30m
```

Key	Value
---	-----
token	s.5YmCHHV80mN3dJpzOwvVAYk8
token_accessor	2ogWa36gDH5wsO8VbuxroByx
token_duration	30m
token_renewable	true
token_policies	["default" "training"]
identity_policies	[]
policies	["default" "training"]

```
$ vault token revoke 2ogWa36gDH5wsO8VbuxroByx  
Success! Revoked token (if it existed)
```



Token Accessors

Renew a Token using the Accessor

Terminal

```
$ vault token renew -accessor gFq2UwnJ0jo87kESKwUcl1Ub
```

Key	Value
---	-----
token	n/a
token_accessor	gFq2UwnJ0jo87kESKwUcl1Ub
token_duration	30m
token_renewable	true
token_policies	["default" "training"]
identity_policies	[]
policies	["default" "training"]

Renewed TTL



Token Accessors

Cannot Use an Accessor to Perform Traditional Vault Actions

Terminal

```
$ vault token create -policy=training -ttl=30m
Key          Value
---          -
token       s.vZRfetFFawRIVKJu8Uc50M9o
token_accessor gFq2UwnJ0jo87kESKwUcl1Ub
token_duration 30m
token_renewable true
token_policies ["default" "training"]
identity_policies []
policies      ["default" "training"]

$ set VAULT_TOKEN=gFq2UwnJ0jo87kESKwUcl1Ub

$ vault kv get secret/apps/training
Error making API request.

URL: GET
http://127.0.0.1:8200/v1/sys/internal/ui/mounts/secret/apps/training
Code: 403. Errors:

* permission denied
```



Explaining Time-To-Live (TTL)

Find A Hotel

Hotel Meetings & Events

Looking for Vacation Packages? [ROOM + FLIGHT](#)  | [ROOM + CAR](#) 



Destination

London, UK

Dates

< Mon, Oct 11 > | < Tue, Oct 19 >

8 NIGHTS

Rooms & Guests

1 Room: 1 Adult/Room

Special Rates

None

Brands

All Brands

Use Points / Certificates

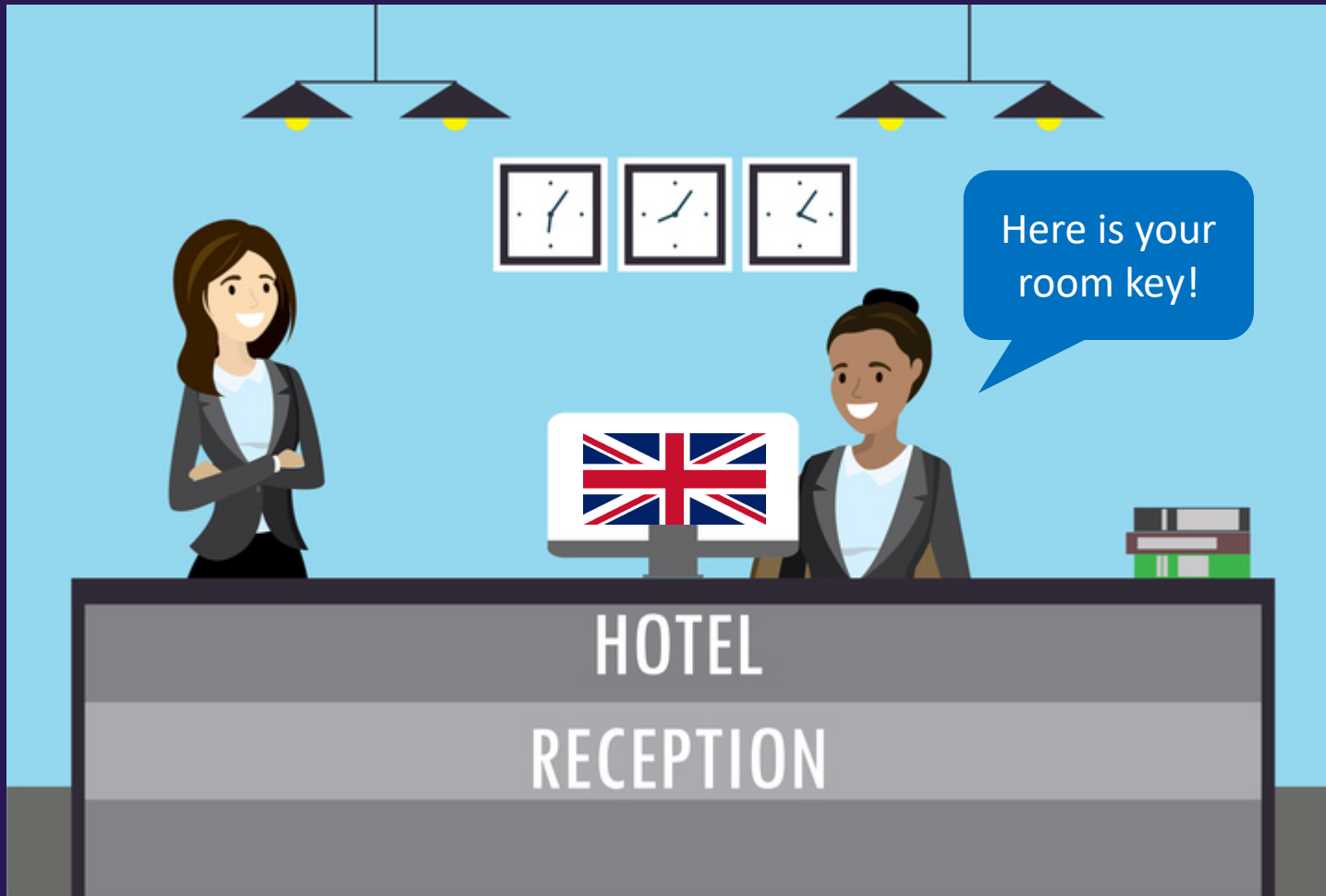
FIND HOTELS



Explaining Time-To-Live (TTL)



VALID FOR 8 DAYS
(TTL)



Time-To-Live (TTL)

Every non-root token has a **TTL**, which is the period of validity (how long it's good for)

TTL is based on the creation (or renewal) time:

- **Example**: New token was created - valid for 30 minutes from now
- **Example**: token was just renewed for 30 min = has a new 30m TTL

When a token's TTL expires, the token is revoked and is no longer valid and cannot be used for authentication.

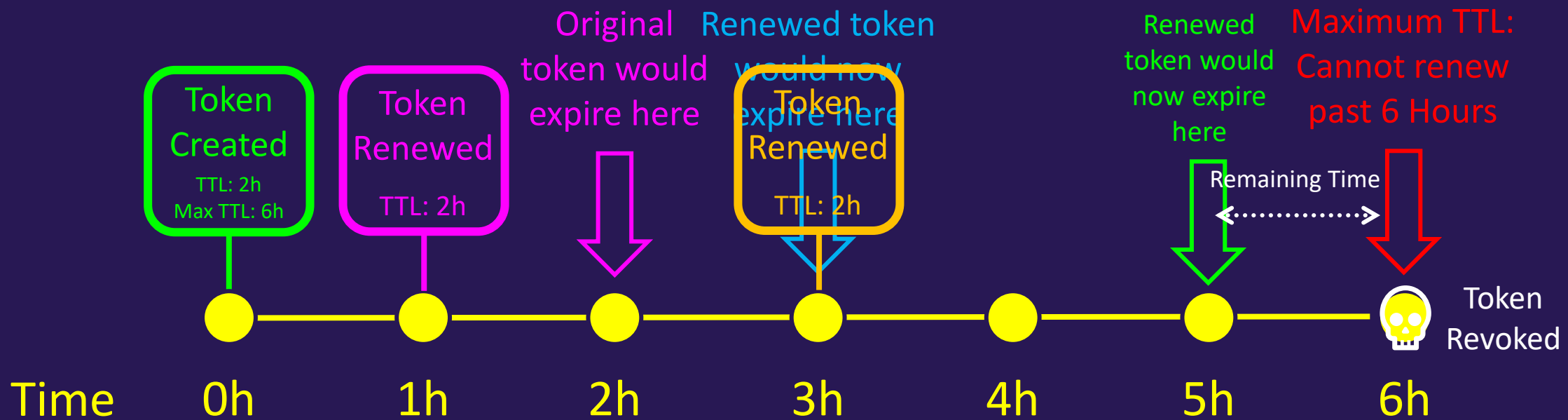
- Renewal must take place before the TTL expires



Time-To-Live (TTL)

A token can have a **TTL** and a **Max TTL**

- This means the token can be renewed up until the Max TTL
- Once the token hits the Max TTL, it cannot be renewed further



Time-To-Live (TTL) Example



←.....→
Still had 3 hours on Max TTL but token is now expired and is no longer valid



Default Time-To-Live (TTL)

Vault has a default TTL of 768 hours (which is 32 days)

- This can be changed in the Vault configuration file
 - `default_lease_ttl = 24h`



How Do I Set the TTL for Tokens?

1. Set the TTL explicitly when creating a token:

```
vault token create -policy=training -ttl=60m
```

2. Configuration of an Auth Method that results in token with a specific TTL

```
vault write auth/approle/role/training-role \  
  token_ttl=1h \  
  token_max_ttl=24h
```

3. The default TTL for Vault will be applied to tokens that are created without explicitly providing a TTL

```
vault token create -policy=training
```



Create a Token Based on Needs

Requirements

1. You have a long-running app which can not handle the regeneration of a token or secret.
2. You need a token that can be renewed indefinitely (forever)

Periodic Token



Create a Token Based on Needs

Requirements

1. You want to limit the token to be used only 3 times regardless of remaining TTL

Service Token with Use Limits



Create a Token Based on Needs

Requirements

1. You need a token that isn't impacted by the lifecycle of its parent
2. You want a token that has an expiration beyond the token that created it

Orphan Token



Create a Token Based on Needs

Requirements

1. You want a token that can only be used by a specific host or within a certain network block

CIDR-Bound Token

(Configured using a Token Role or Different Auth Methods)

P.S - Don't let this confuse you – it's basically just a regular service token with additional CIDR-bound configuration



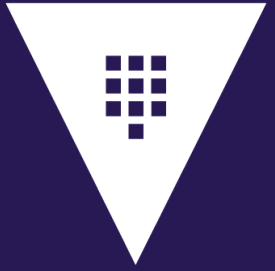
Create a Token Based on Needs

Requirements

1. You need a token that is replicated to all other Vault clusters in a replica set
2. You need to reduce/minimize the overhead on your storage backend when creating a large number of tokens

Batch Token





Exam Tips for Objective 3



Exam Tips

- Know the different types of tokens
 - Service Token
 - Batch Token
 - Root Token
 - Periodic Token
 - Orphan Token
 - CIDR-Bound Token
- Remember what each token is and what makes it unique. Don't forget what use cases would merit the use of each type.



Exam Tips

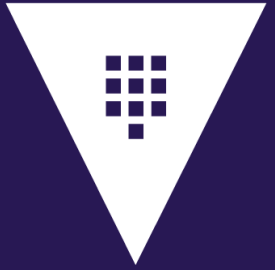
- Know the differences between a service token and a batch token
 - Refer to the comparison chart in "Comparing Token Types"
- Remember that **service tokens are written to storage** and batch tokens are **not**
- Practice using the **vault token** command and the different options and flags available



Exam Tips

- Remember the different ways to create a root token
- Know that the root token should be revoked after being used, the root token does not expire, and it allows unrestricted access to Vault.
- Don't forget the (4) actions that you can perform with a token accessor
- Remember the default TTL in Vault is **768h** (32 days)





END OF SECTION

