Securely Configure Auto-Auth and Token Sink

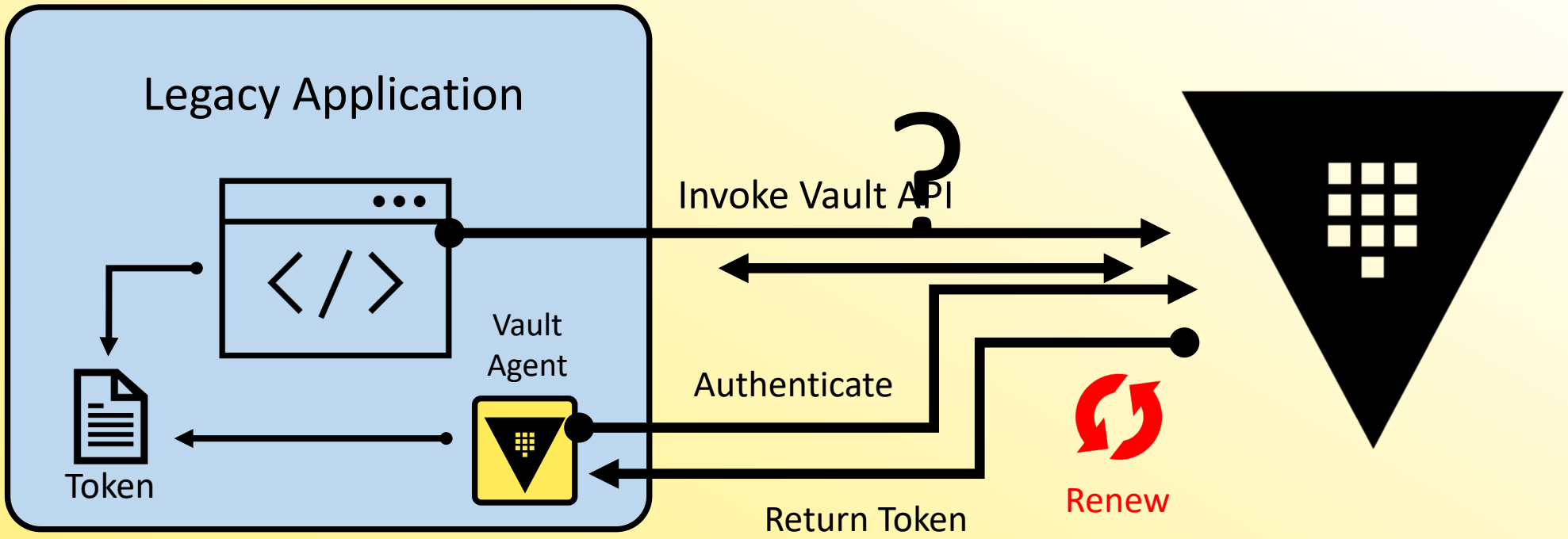
What is the Vault Agent?

The Vault Agent is a client daemon that runs alongside an application to enable legacy applications to interact and consume secrets

Vault Agent provides several different features:

- Automatic authentication including renewal
- Secure delivery/storage of tokens (response wrapping)
- Local caching of secrets
- Templating

Legacy Applications – Auto-Auth



Vault Agent – Auto-Auth

- The Vault Agent uses a pre-defined auth method to authenticate to Vault and obtain a token
- The token is stored in the "sink", which is essentially just a flat file on a local file system that contains the Vault token
- The application can read this token and invoke the Vault API directly
- This strategy allows the Vault Agent to manage the token and guarantee a valid token is always available to the application

Vault Agent - Auto Auth

Vault Agent supports many types of auth methods to authenticate and obtain a token

Auth methods are generally the methods you'd associate with "machine-oriented" auth methods

- **`AliCloud`**
- **`AppRole`**
- **`AWS`**
- **`Azure`**
- **`Certificate`**
- **`CloudFoundary`**
- **`GCP`**
- **`JWT`**
- **`Kerberos`**
- **`Kubernetes`**

Vault Agent Configuration File

```
auto_auth {
  method "approle" {
      mount_path = "auth/approle"
      config = {
          role_id_file_path = "<path-to-file>"
          secret_id_file_path = "<path-to-file>"
      }
  }

  sink "file" {
      config = {
          path = "/etc/vault.d/token.txt"
      }
  }
}

vault {
  address = "http://<cluster_IP>:8200"
}
```

Auto-Auth Configuration (AppRole)

Sink Configuration

Vault

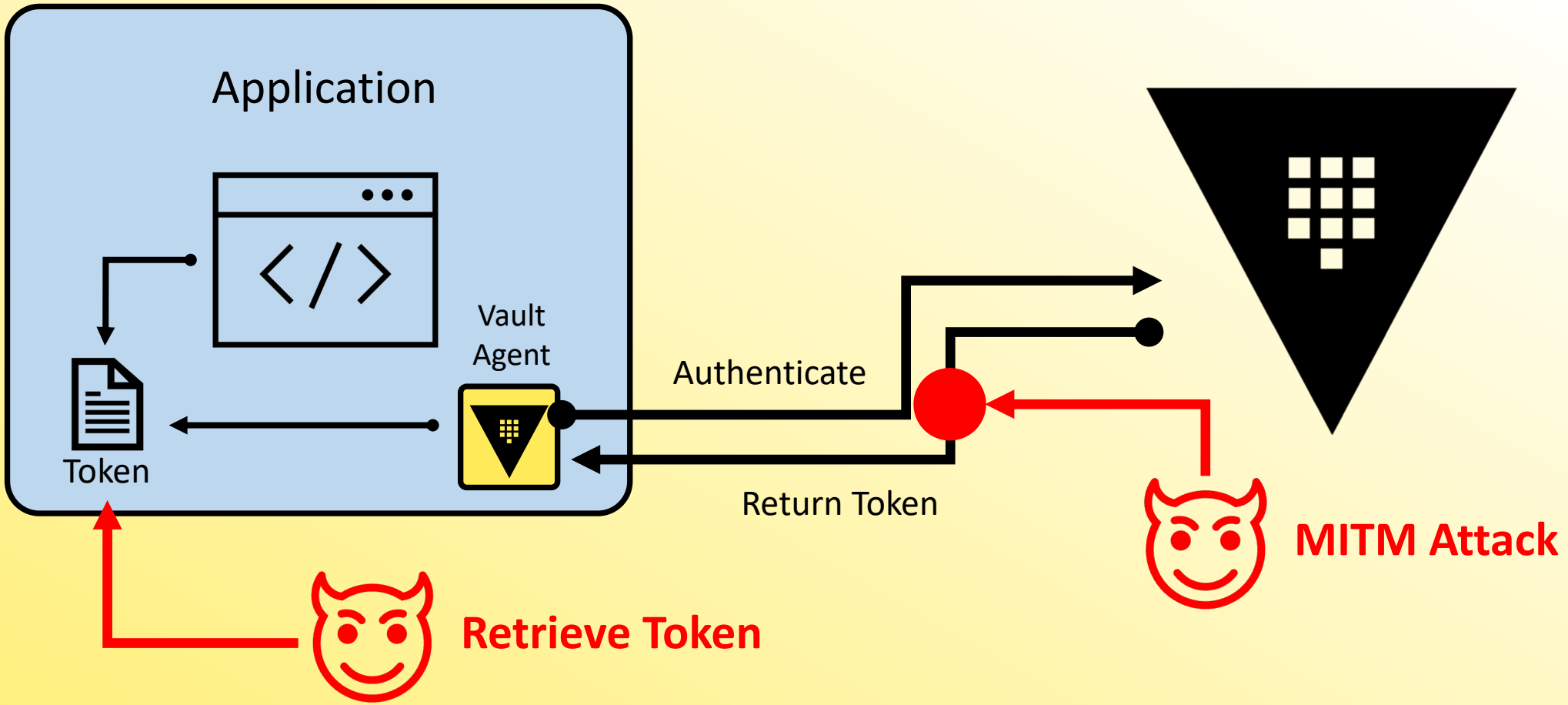
Vault Agent - Sink

As of today, `file` is the only supported method of storing the auto-auth token

Common configuration parameters include:

- `type` – what type of sink to use (again, only `file` is available)
- `path` – location of the file
- `mode` – change the file permissions of the sink file (default is 640)
- `wrap_ttl` = retrieve the token using response wrapping

Auth-Auth Security Concerns



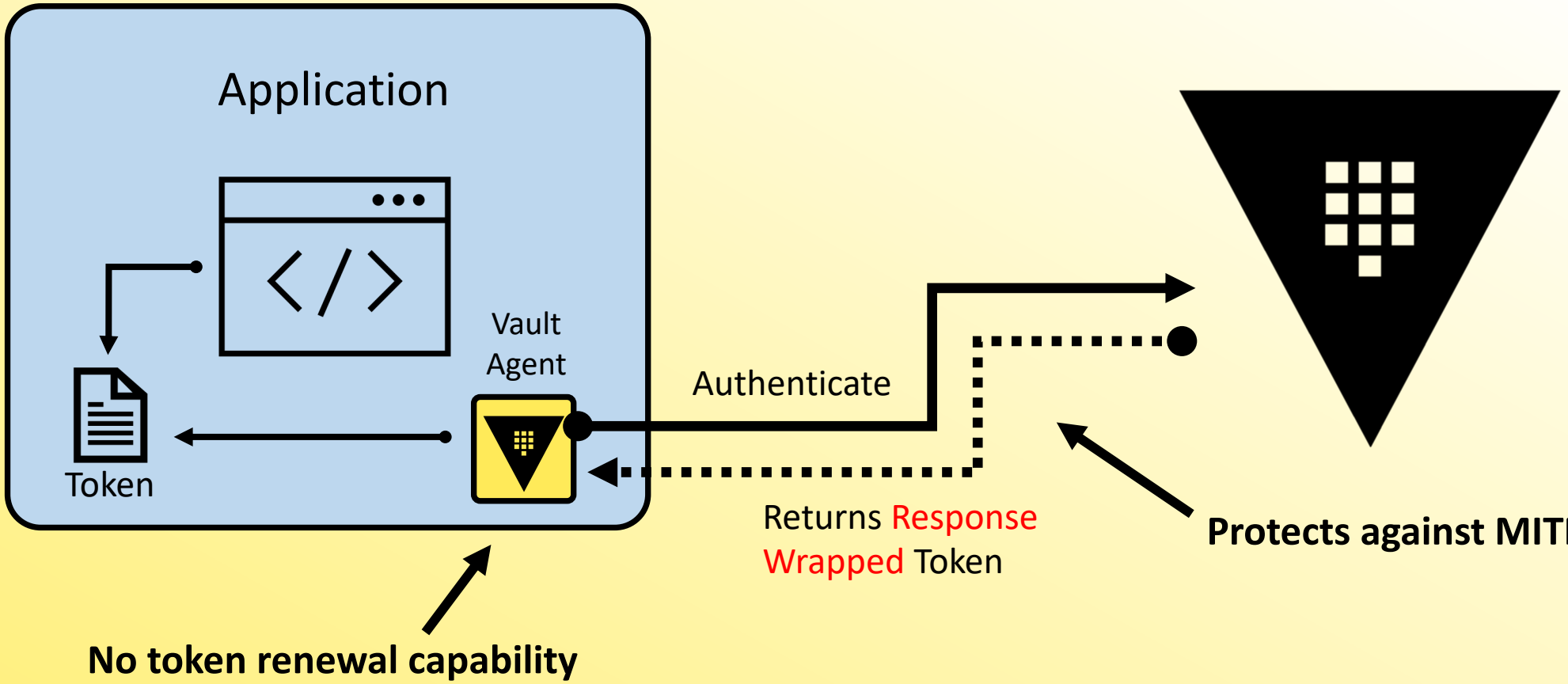
Protecting the Token using Response Wrapping

To help secure tokens when using Auth-Auth, you can have Vault response wrap the token when the Vault Agent authenticates

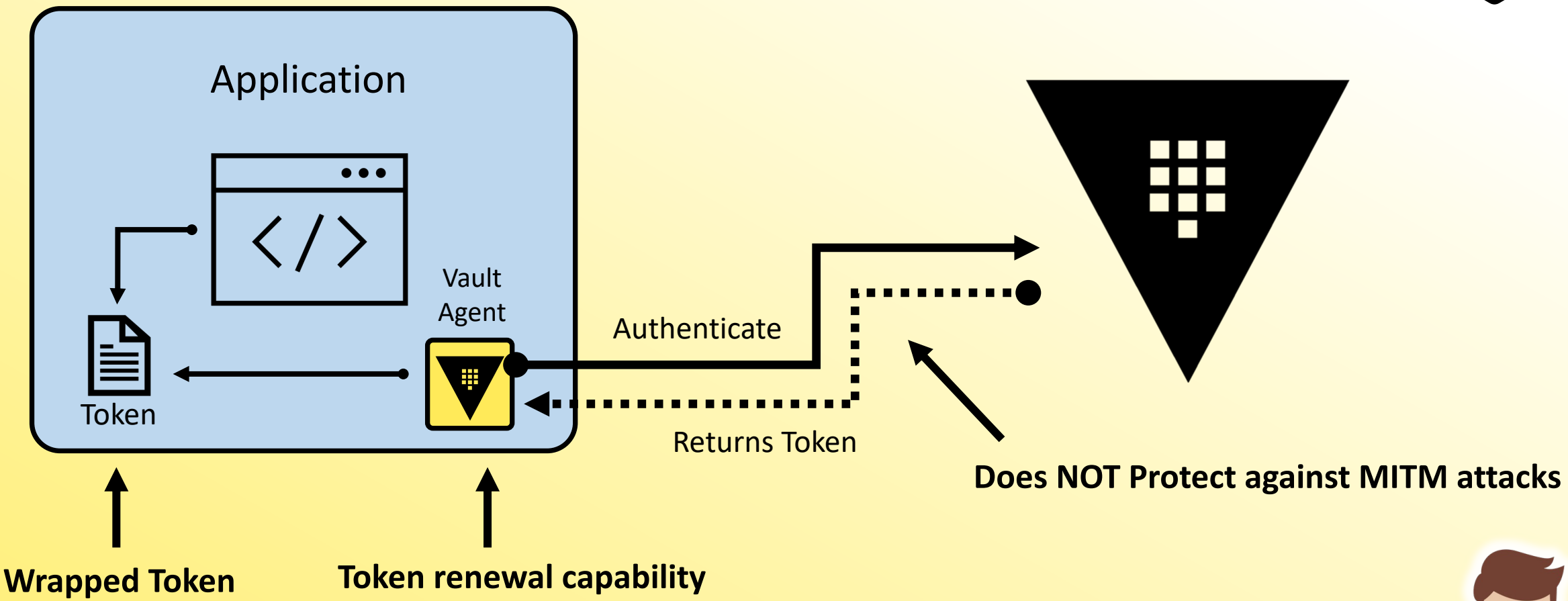
- Response wrapped by the auth method
- Response wrapped by the token sink

The placement of the **`wrap_ttl`** in the Vault Agent configuration file determines where the response wrapping happens.

Response Wrapping at the **Auth Method**



Response Wrapping at the **Sink**



Response-Wrapping the Token - Comparison

Response Wrapped by the Auth Method

Pros:

- Prevents man-in-the-middle attacks (MITM)
- More secure

Cons:

- Vault agent cannot renew the token

Response Wrapped by the Sink

Pros:

- Allows the Vault agent to renew the token and re-authenticate when the token expires

Cons:

- The token gets wrapped ***after*** it is retrieved from Vault. Therefore, it is vulnerable to MITM attack

Response-Wrapping the Token - Comparison

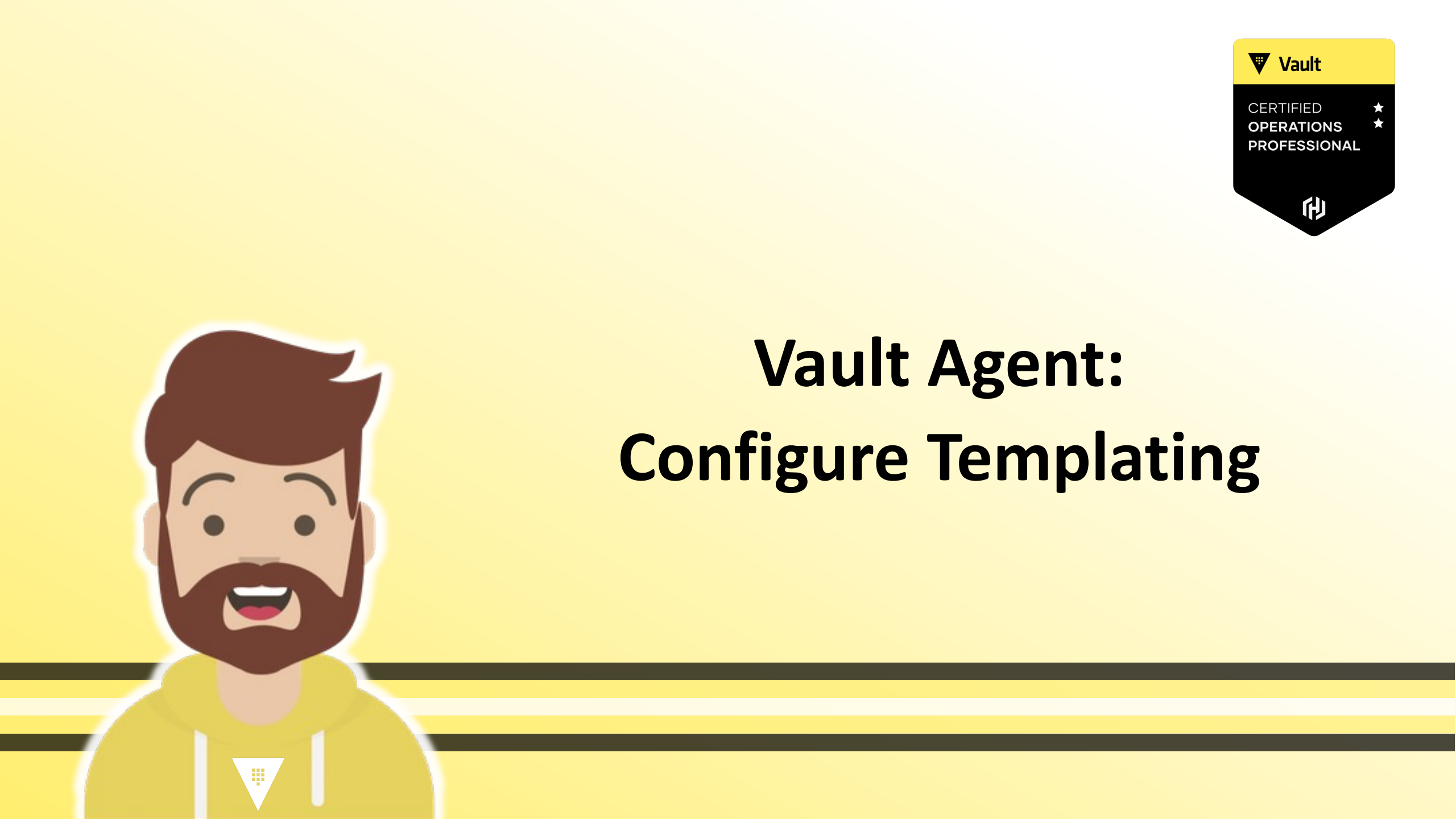
Response Wrapped by the Auth Method

```
pid_file = "/home/vault/pidfile"

auto_auth {
   method "kubernetes" {
     wrap_ttl = "5m"
     mount_path = "auth/kubernetes"
     config = {
       role = "example"
     }
   }
...

vault {
  address = "http://<cluster_IP>:8200"
}
```

Response Wrapped by the Sink

```
pid_file = "/home/vault/pidfile"

auto_auth {
  method "kubernetes" {
    mount_path = "auth/kubernetes"
     config = {
       role = "example"
     }
  }
   sink "file" {
      wrap_ttl = "5m"
      config = {
        path = "/etc/vault/token"
      }
   }
}
```

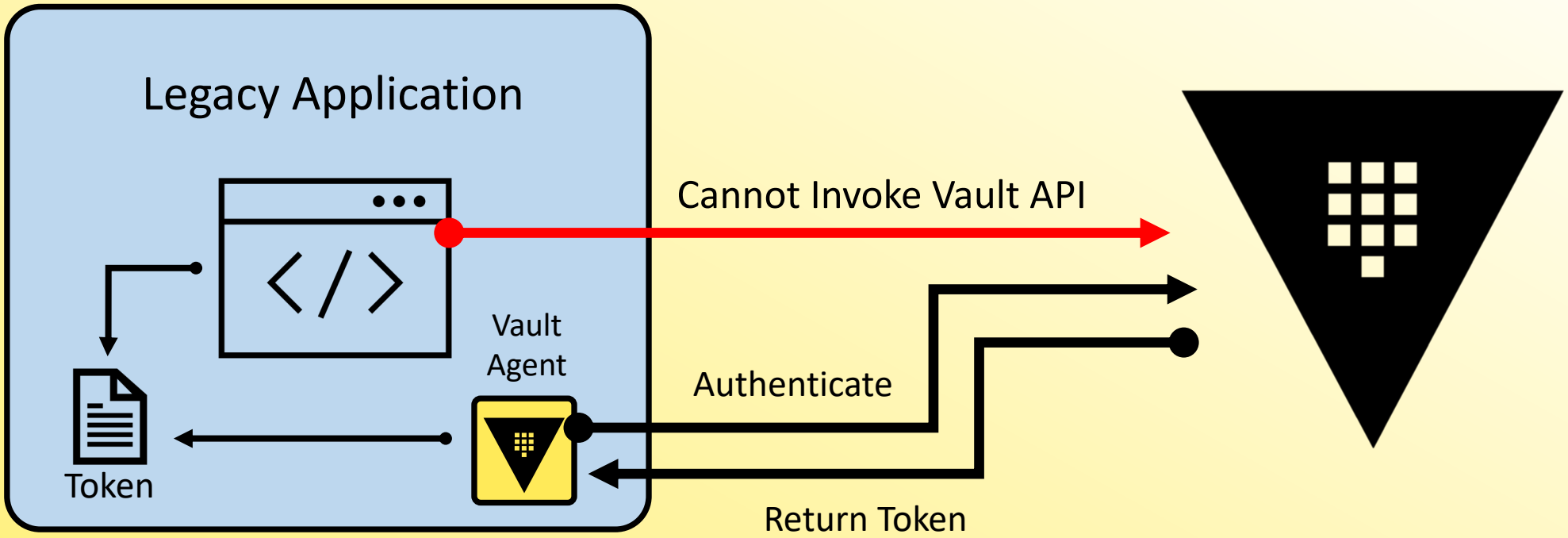
END OF SECTION



Vault Agent: Configure Templating

Legacy Applications – Auto-Auth

How can legacy applications still take advantage of Vault for secrets?



Consul Template

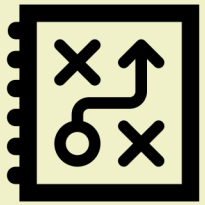
- A standalone application that renders data from Consul or Vault onto the target file system
  - https://github.com/hashicorp/consul-template
  - Despite its name, Consul Template does **not** require a Consul cluster to operate
- Consul Template retrieves secrets from Vault
  - Manages the acquisition and renewal lifecycle
  - **Requires a valid Vault token to operate**

Consul Template - Workflow

Step 1

Create a templated file that specifies the path and data you want

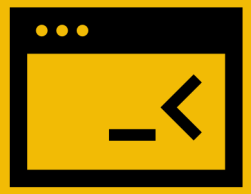
Step 2



Execute Consul Template

Consul Template retrieves data and renders to destination file

Step 3



The application reads the new file at runtime that includes secrets retrieved from Vault

Consul Template

Create a templated file

```
production:
  adapter: postgresql
  encoding: unicode
  database: orders
  host: postgres.hcvop.com
  {{ with secret "database/creds/readonly" }}
  username: "{{ .Data.username }}"
  password: "{{ .Data.password }}"
  {{ end }}
```

The path in Vault to read secrets from

The data to be retrieved and consumed from the Vault response

Consul Template – Destination File (Result)

Destination File that application server will read at runtime

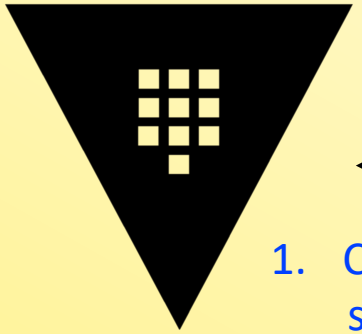
```
production:
  adapter: postgresql
  encoding: unicode
  database: orders
  host: postgres.hcvop.com
  username: "v-vault-readonly-fm3dfm20sm2s"
  password: "fjk39fkj49fks02k_3ks02mdz1s1"
```

Consul Template - Workflow

CONSUL TEMPLATE

INPUT

```
production:
  adapter: postgresql
  encoding: unicode
  database: orders
  host: postgres.hcvop.com
  {{ with secret "database/creds/readonly" }}
  username: "{{ .Data.username }}"
  password: "{{ .Data.password }}"
  {{ end }}
```

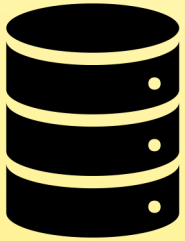


1. Consul Template gets secrets from Vault

2. Secrets are rendered to the targeted output file

OUTPUT

```
production:
  adapter: postgresql
  encoding: unicode
  database: orders
  host: postgres.hcvop.com
  username: "v-vault-readonly-fm3dfm20sm2s"
  password: "fjk39fkj49fks02k_3ks02mdz1s1"
```

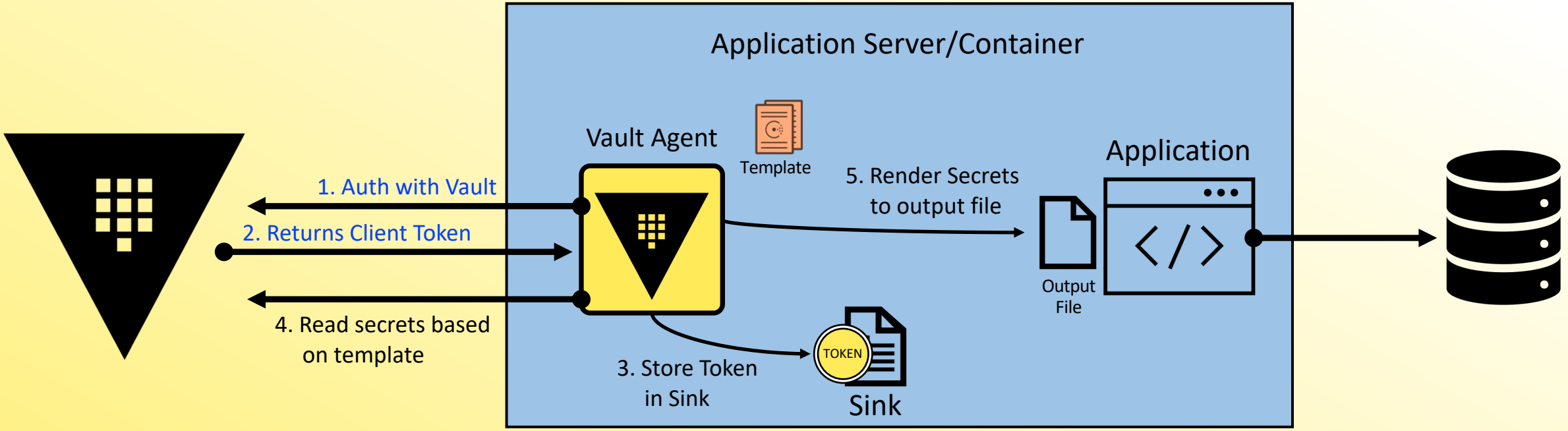


3. The application reads the file at runtime and access the database

Vault Agent Templating

- To further extend the functionality of the Vault Agent, a subset of the Consul-Template functionality is directly embedded into the Vault Agent
  - No need to install the Consul-Template binary on the application server
- Vault secrets can be rendered to destination file(s) using the Consul-Template markup language
  - Uses the client token acquired by the auto-auth configuration

Vault Agent Templating



Template Configuration

```
auto_auth {
  method "approle" {
     mount_path = "auth/approle"
  ...

  sink "file" {
     config = {
         path = "/etc/vault.d/token.txt"
  ...

template_config {
  exit_on_retry_failure = true
  static_secret_render_interval = "10m"
}

template {
  source = "/etc/vault/web.tmpl"
  destination = "/etc/webapp/config.yml"
}
```

Auto-Auth Configuration (AppRole)

Sink Configuration

Global Template Configurations
(affects all templates)

Template Configuration

END OF SECTION