# Using External Secrets Operator (ESO) in epic-app to inject secrets



## Document Level Classification

200

# Introduction

When developing and deploying services on the Cloud Developer Platform in a Substrate account, your service may need to access other backend or downstream systems.  In order to access these systems, sensitive information is necessary to authenticate with them. This could be a password, token, API key, etc. In order to use this sensitive information in a service, it must be done securely.  Since services in Substrate are deployed on the Kubernetes platform, Kubernetes Secrets are used so sensitive information is not exposed in application code.

# What is External Secrets Operator (ESO)?

From the official documentation, External Secrets Operator is a Kubernetes operator that integrates external secret management systems like AWS Secrets Manager, HashiCorp Vault, Google Secrets Manager,

Azure Key Vault, IBM Cloud Secrets Manager, CyberArk Conjur and many more. The operator reads information from external APIs and automatically injects the values into a Kubernetes Secret.

On the Epic Cloud Developer Platform in Substrate accounts, we will be focused on the use of ESO with HashiCorp Vault provider. Epic's implementation of HashiCorp Vault is referred to as Substrate Vault.

# What are the benefits of ESO?

- Offers the ability to use native Kubernetes secrets.
- Easy authentication and authorization access from a given cluster simply by placing the secret in the right place in the vault path.
- No longer need to manually share secrets using SSSM as this is handled via automation.
- No entry point scripts that can cause SIGTERM trap problems.
- Integration in the epic-app Helm chart supports **ExternalSecret** resources that users can define in a simplified form on their values.yaml files.

# Why ESO?

We understand that many of our users had found the Vault Injector and SSSM challenging due to their operational complexity and dependence on outdated concepts like TONUAM groups. ESO offers a simpler, vendor-neutral solution that reduces friction and eliminates the need for future migrations.

# How does ESO differ from previous solutions?

The old Substrate secret consumption mechanism relies on many tools such as SSSM, Starmap and others. All of these tools increase the complexity and the amount of manual operations. It also does not allow standard use of Kubernetes Secret features (such as the ability to reference secrets from workload container environment variable specifications), requiring additional support be implemented in an application. Finally the existing Substrate secrets consumption mechanism, the Vault Injector, requires Vault to be available whenever new pods are scheduled, which reduces resilience.

ESO allows Substrate customers to use ESO ExternalSecret resources to consume Kubernetes secrets in the standard fashion with per-namespace authorization. With ESO the amount of manual provisioning, when clusters, namespaces, workloads or secrets come and go, is eliminated because ESO doesn't use any of the legacy tools/systems such as SSSM. Finally, the impact of Vault availability on the reliability of horizontal scaling is eliminated, meaning if Vault goes down, the synced secrets will still remain in a cluster.

# Am I obliged to use ESO?

Starting from August 2024, all new Kubernetes clusters use ESO. Vault injector is depreciated but would not be forcefully removed.

# How do I use ESO in my Substrate application?

ESO can be enabled in epic-app by setting the appropriate values in your values.yaml file. You need at least epic-app version **3.5.0+** to be able to use its ESO integration. You may find the epic-app release notes here and the upgrade documentation here.

In the following example we are enabling ESO in epic-app by setting `enabled: true` in the `externalSecrets` section. Under `env` we are setting 2 environment variables called `PGUSER` and `PGPASSWORD` to values of secrets in Vault that are specified by their path in `key:` and the name of the secret in Vault specified by `property:`. Variables under `externalSecrets.env` are injected as environment variables into every container in the Deployment. This assumes the namespace in the Vault path matches the namespace in which the containers are deployed, as access to secrets is on a per namespace basis.

```
externalSecrets:
  enabled: true
  env:
    PGUSER:
      key: <cluster-name>/<namespace-name>/<secret>
      property: username
    PGPASSWORD:
      key: <cluster-name>/<namespace-name>/<secret>
      property: password
      version: '1'
  refreshInterval: 12h
  deletionPolicy: Retain
```

In the following example we are enabling ESO in epic-app by setting `enabled: true` in the `externalSecrets` section. Next we are setting `containers.substrate-example.externalSecrets` with values. This will inject secrets as environment variables into ONLY these containers in the namespace. This assumes the namespace in the Vault path matches the namespace in which the containers are deployed, as access to secrets is on a per namespace basis.

```
externalSecrets:
  enabled: true
  refreshInterval: 12h
  deletionPolicy: Retain

containers:
```

```
  substrate-example:
    image:
      name: "artifacts.ol.epicgames.net/<DOCKER_REPO>/substrate-example
      tag: "1.0.0-<IMAGE_NAME>"
    ports:
      - 80
    environment:
      LOG_LEVEL: INFO
    externalSecrets:
      PGUSER:
        key: <cluster-name>/<namespace-name>/<secret>
        property: username
      PGPASSWORD:
        key: <cluster-name>/<namespace-name>/<secret>
        property: password
        version: '1'
```

# How can I map multiple secrets at once?

The below example shows how you can map multiple secrets at once when there are multiple values in the path. This is for when you have many secrets and you don't want to manually define the mappings one at a time. When mounted on a container these secrets are referenceable as environmental variables.  For instance if there are three secrets in the path called key1, key2, and key3; you can reference their values via their environmental variable names which would also be key1, key2, and key3.

Note: This feature was added as of version **3.6.0.** You may find the epic-app release notes here and the upgrade documentation here.

```
externalSecrets:
  enabled: true
  templated:
    - name: many-secrets
```

```
      dataFrom:
        - extract:
            key: <cluster-name>/<namespace-name>/<multi-secret-path>
            version: "1"
containers:
  substrate-example:
    image:
      name: "artifacts.ol.epicgames.net/<DOCKER_REPO>/substrate-example
      tag: "1.0.0-<IMAGE_NAME>"
    ports:
      - 80
    environment:
      LOG_LEVEL: INFO
    environmentValueFromResource:
        - resourceType: secretRef
          resourceName: "many-secrets"
```

# What are the supported Vault paths?

In Substrate Kubernetes Clusters, there are 2 supported vault paths for use with ESO.

- Namespace Bound Vault Path
- All Namespaces Vault Path

## Namespace Bound Vault Path

With this path, the Vault path is bound to a specific namespace on the cluster. Instead of having all secrets under the same root path `/secret` we decided to use root paths that are unique for each AWS account, e.g. `/abcd-dev`. If you have access to at least one Kubernetes cluster, you will see additional path(s) upon logging in to the Substrate Vault. The complete path follows the format `/<account-name>/<cluster-name>/<namespace-name>`.

## Namespace Bound Vault Path Example

| | |
|---|---|
| **Account Name** | dead-dev |
| **Cluster Name** | dead-dev-substrate |
| **Namespace** | team-online-infra-platform |
| **Secret** | postgresql |
| **Resultant Vault Path** | **/dead-dev/dead-dev-substrate/team-onlin** |

## All Namespaces Vault Path

With this path, the vault path is not bound to a specific namespace on the cluster. If a secret is placed in this path, all Kubernetes namespaces in a cluster can read it. This is to avoid copying common secrets (e.g. observability API keys, etc) across multiple locations. The complete path follows the format `/<account-name>/<cluster-name>/ALL_NAMESPACES`.

## All Namespaces Vault Path Example

| | |
|---|---|
| **Account Name** | dead-dev |
| **Cluster Name** | dead-dev-substrate |

| | |
|---|---|
| **Namespace** | ALL_NAMESPACES |
| **Secret** | postgresql |
| **Resultant Vault Path** | **/dead-dev/dead-dev-substrate/ALL_NAMES** |

# Do I need to request any access?

No. Everyone who has access to a given Kubernetes cluster automatically gets access to this cluster's path in the Substrate Vault. The only exception here is people with global Kubernetes access. If this is the case for you, please request access to the desired cluster(s) via K8S-SSO application in Sailpoint.

# How do I create secrets for use with ESO?

Secrets are created in Substrate Vault. To create new secrets in Substrate Vault, perform the following:

1. Sign-in to Vault (using Okta credentials)
2. Navigate to the secrets location for your account, cluster, and namespace. For example `/<account-name>/<cluster-name>/<namespace-name>/<secret>` .
3. Click **Create secret +.**
4. Enter a name for the secret in the **Key** field and the secret data in the **Value** field next to it.
5. Click the **Save** button to save the secret.

# How do I move secrets from an existing Vault path to an ESO Vault path?

Currently you cannot explicitly move secrets from one path to another due to Vault limitation. You will have to recreate your secrets that you created for use with Vault Injector to the new path to use them with ESO. For example, if you previously used a secret path / `secret/my-brand/my-project/use1a/live/runtime/database-creds` , the secrets that exist under the `database-creds` path would need to be recreated in ESO path that correspond to the account, cluster, and namespace that those secrets will be made available to. That new path would look something like `/dead-dev/dead-dev-substrate/team-online-infra-platform/database-creds` .

## Moving Secrets with Vault CLI

You can move your secrets using the vault CLI by downloading your secrets from their current path to a json file, then creating them in the new path from the json file by using the following directions.

### Old Prod Vault to Substrate Vault

1. Download vault cli from https://releases.hashicorp.com/vault/ and place the executable somewhere in your $PATH.

2. Open a terminal tab

3. Login to OldProd Vault

**Mosaic** macros cannot be exported to this format.

**Mosaic** macros cannot be exported to this format.

4. Copy secrets from OldProd

```
vault kv get -format=json /source/path | jq .data.data > vault.json
```

5. Open a new Terminal tab

6. Login to Substrate Vault

**Mosaic** macros cannot be exported to this format.

**Mosaic** macros cannot be exported to this format.

7. Copy secrets to Substrate

```
vault kv put /destination/path @vault.json
```

## Substrate Vault to ESO path for Substrate Vault

1. Download vault cli from https://releases.hashicorp.com/vault/ and place the executable somewhere in your $PATH.

2. Open a terminal tab

3. Login to Substrate Vault

**Mosaic** macros cannot be exported to this format.

**Mosaic** macros cannot be exported to this format.

4. Copy secrets from one path to another.

```
vault kv get -format=json /source/path | jq .data.data > vault.json | v
```

# How do I use my secrets in epic-app?

In order to utilize the secrets using epic-app, you would also need to update your code to enable externalSecrets and remove the podAnnotation that were used inject secrets using Vault Injector.

**Mosaic** macros cannot be exported to this format.

**Mosaic** macros cannot be exported to this format.

**Mosaic** macros cannot be exported to this format.

**Mosaic** macros cannot be exported to this format.

# Can I use ESO if I am not using epic-app?

Yes, if you would like to use ESO in your application but you are not using epic-app, you would need to define an ExternalSecret in your Helm chart. See the example below.

```
apiVersion: external-secrets.io/v1
kind: ExternalSecret
metadata:
  name: <some_name>
spec:
  refreshInterval: "15s"
  secretStoreRef:
    name: eso-vault
    kind: SecretStore
  target:
    name: <name_of_the_target_k8s_secret>
  data:
  - secretKey: <key_name_in_target_k8s_secret>
```

```
    remoteRef:
      key: <vault_path> # for example "adcc-live-unreal-code-scout/ALL_
      property: <vault_secret_key>
```

After you have defined the ExternalSecret, you would then reference it as you would a Kubernetes Secret. Examples can be found in the official documentation for Using a Secret.

# Is there any automation I can use to move my secrets to the ESO Vault path?

No, the process is manual at this moment.

# Can I use anything other than Vault with ESO?

Our ESO solution currently supports only Substrate Vault. We welcome your feedback to determine if there's a need to support additional secret management systems. Reach us out in #cloud-ops-support-ext.

# Can I use the new layout with secrets that don’t belong to any Kubernetes clusters?

Not yet, but we are working on it.

# I have a question that is not answered here

Please let us know about your question in the #cloud-ops-support-ext.

---