# Prueba de concepto de ArgoCD Vault Plugin (AVP)

* [Prueba de concepto de instalación de ArgoCD Vault Plugin (AVP)](#README_md).
* [Helm y Go](#10_helm_go_md) .
* [Instalación de Vault según la documentación de OpenShift](#11_vault_fail_md).
* [Instalación de Vault](#11_vault_md).
* [Instalación de ArgoCD](#12_argocd_md).
* [ArgoCD Vault Plugin (AVP)](#20_avp_md).
* [Instalación de ArgoCD Vault Plugin (AVP)](#21_avp_install_md).
* [Autenticación y autorización por approle en Vault](#22_avp_vault_approle_md).
* [Despliegue de aplicaciones en ArgoCD con Vault approle](#23_avp_argocd_approle_md).

# Prueba de concepto de instalación de ArgoCD Vault Plugin (AVP)

## Resumen de ArgoCD Vault Plugin (AVP)

ArgoCD Vault Plugin (AVP) es un ejecutable de Linux que ArgoCD invoca antes  
de desplegar un manifiesto de un objeto en OpenShift  
para sustituir en el manifiesto unas marcas especiales  
por valores secretos almacenados en un sistema de gestión de secretos.

Típicamente este manifiesto a modificar será el de un Secret (también un ConfigMap) con este aspecto:

kind: Secret

apiVersion: v1

metadata:

name: example-secret

annotations:

avp.kubernetes.io/path: "avp/data/test"

type: Opaque

stringData:

sample-secret: <sample>

La anotación avp.kubernetes.io/path: indica que el manifiesto está sujeto a ser modificado por ArgoCD Vault Plugin (AVP)  
con los valores que un sistema externo almacena en un path interno de ese sistema "avp/data/test".

El sentido del path en ese sistema depende del sistema que se configure, y aplica por igual a todos los manifiestos que esa instancia de ArgoCD maneje con AVP.

En última instancia, en esta prueba de concepto se consigue que el código fuente de la aplicación no tenga almacenado ningún secreto.

El aspecto a mejorar, es que una vez desplegado es secreto este queda visible para los administradores de cluster y parar los administradores del namespace donde reside.  
Ésto último se puede mejorar al coste de una mayor complicación.

## Objetivo de esta prueba piloto

La prueba piloto se divide en dos grandes áreas:

* Instalación de una infraestructura en OpenShift que facilite la prueba piloto en si.
* Las operaciones de configuración de la prueba piloto que permiten desplegar una aplicación con un Secret donde AVP pondrá el valor de un secreto externo en la marca prevista.

### Infraestructura

La infraestructura a instalar es:

* [Helm y Go](" \l "10_helm_go_md) como prerrequisitos para poder instalar ArgoCD.
* [Vault](#11_vault_md) como sistema externo que gestionará los secretos.
* [ArgoCD](#12_argocd_md) como Custom Resource Definition (CRD) lo que facilita la configuración respecto al Operador ofrecido por OpenShift, más estricto a la hora de hacer cambios.
* [ArgoCD Vault Plugin (AVP)](#20_avp_md) como lazo final.

### Configuración y prueba

La configuración más compleja es definir como el ArgoCD Vault Plugin (AVP) es capaz de presentarse  
ante el sistema externo, demostrar su identidad (autenticación/authn) y a partir de estar lograr acceso para leer (autorización/authz) el valor que ha de escribir en la marca del manifiesto que ha de modificar antes de su despliegue.

Para authn y authz del AVP hay dos opciones consideradas en esta prueba, se optó por la primera.

* [Approle de Vault](file:///C:\_exe\E31710\Fuentes\vault\001.protoclusterTRT\tutorials\30-approle.md), en que se usan "cuentas" definidas en Vault, los role\_id y el equivalente de sus constraseñas, los secret\_id.
* [Service account de Openshift](file:///C:\_exe\E31710\Fuentes\vault\001.protoclusterTRT\tutorials\31-sa-rbac.md), en que se usan las cuentas de servicio de Openshift y la autorización RBAC (Orle Based Access Control) de OpenShift.

Una vez configurado authn y authz, el sistema externo puede interpretar el path  
(o valor dado en la anotación para que AVP actúe) y permitir que AVP consulte el secreto para hacer la sustitución en el manifiesto.

El último paso es [crear una aplicación en ArgoCD](file:///C:\_exe\E31710\Fuentes\vault\001.protoclusterTRT\tutorials\40-app.md) y ver como efectivamente se hace la sustitución.

### CyberArk

De momento no se considera en vistas de los requerimientos expresados:

* <https://docs.cyberark.com/AAM-DAP/12.4/en/Content/Integrations/k8s-ocp/k8s_lp.htm>

# Helm y Go

Helm y Go son prerrequisitos para poder instalar ArgoCD.

ArgoCD se instalará como Custom Resource Definition (CRD) lo que facilita la configuración respecto al Operador ofrecido por OpenShift, más estricto a la hora de hacer cambios.

## Go

El siguiente documento "Installing Helm" indica los prerrequisitos y los pasos necesarios desde la propia consola OpenShift de la línea de comandos (CLI).

<https://docs.openshift.com/container-platform/4.10/applications/working_with_helm_charts/installing-helm.html>

*Prerequisites: You have installed Go, version 1.13 or higher.*

En nuestro escritorio de preferencia, en este ejemplo Windows,  
instalamos la versión 1.20.5 Go go1.20.5.windows-amd64.msi desde <https://go.dev/doc/install>

Validamos la accesibilidad de Go:

go version

go version go1.20.5 windows/amd64

Hemos instalado Go 1.20.5 > 1.13.

## Helm

### Instalar Helm desde páginas de Red Hat

El mismo documento "Installing Helm" indica los prerrequisitos y los pasos necesarios desde la propia consola OpenShift de la línea de comandos (CLI).

<https://docs.openshift.com/container-platform/4.10/applications/working_with_helm_charts/installing-helm.html>

*You can also find the URL to the latest binaries from the OpenShift Container Platform web console by clicking the ? icon in the upper-right corner and selecting Command Line Tools.*

* <https://console-openshift-console.apps.k8spro.nextret.net/command-line-tools>
* <https://mirror.openshift.com/pub/openshift-v4/clients/helm/latest>

### Instalación alternativa desde páginas de Helm

En el momento de hacer la prueba no funciona http://developers.redhat.com/ raíz de las decargas propuetas.  
Tomamos de forma genérica la página de instalación de Helm propiamente dicha <https://helm.sh/docs/intro/install/> que sugiere usar la instalación comunitaria usando Chocolatey...

Dado que se disponía de **Chocolatey**,se hizo la instalación con esta herramienta y como administrador local.

choco install kubernetes-helm

Chocolatey v1.3.0

Installing the following packages:

kubernetes-helm

By installing, you accept licenses for the packages.

Progress: Downloading kubernetes-helm 3.11.3... 100%

kubernetes-helm v3.11.3 [Approved]

kubernetes-helm package files install completed. Performing other installation steps.

The package kubernetes-helm wants to run 'chocolateyInstall.ps1'.

Note: If you don't run this script, the installation will fail.

Note: To confirm automatically next time, use '-y' or consider:

choco feature enable -n allowGlobalConfirmation

Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): Y

Downloading kubernetes-helm 64 bit

from 'https://get.helm.sh/helm-v3.11.3-windows-amd64.zip'

Progress: 100% - Completed download of C:\Users\Admin\AppData\Local\Temp\chocolatey\kubernetes-helm\3.11.3\helm-v3.11.3-windows-amd64.zip (14.91 MB).

Download of helm-v3.11.3-windows-amd64.zip (14.91 MB) completed.

Hashes match.

Extracting C:\Users\Admin\AppData\Local\Temp\chocolatey\kubernetes-helm\3.11.3\helm-v3.11.3-windows-amd64.zip to C:\ProgramData\chocolatey\lib\kubernetes-helm\tools...

C:\ProgramData\chocolatey\lib\kubernetes-helm\tools

ShimGen has successfully created a shim for helm.exe

The install of kubernetes-helm was successful.

Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-helm\tools'

Chocolatey installed 1/1 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Did you know the proceeds of Pro (and some proceeds from other

licensed editions) go into bettering the community infrastructure?

Your support ensures an active community, keeps Chocolatey tip-top,

plus it nets you some awesome features!

https://chocolatey.org/compare

Verificando

helm version

version.BuildInfo {

Version:"v3.11.3",

GitCommit:"323249351482b3bbfc9f5004f65d400aa70f9ae7",

GitTreeState:"clean",

GoVersion:"go1.20.3"

}

Parece que Helm instaló su propio Go 1.20.3 (casi tan al dia como el 1.20.5 anterior, que ahora parece innecesario). La versión 3.11.3 es algo más actual que la 3.11.1 del mirror de Red Hat.

# Instalación de Vault según la documentación de OpenShift

La instalación de Vault según la documentación de OpenShift presenta algunas restricciones tal como se ha seguido:

* No crea la ruta de acceso al servicio que accede a Vault.
* No parece haber instalado la interfaz gráfica de usuario.

Por estos motivos se descarta, si bien interesa observar lo que ha instalado en example-vault.

* Persistent volume claim data-example-vault-0.
* Servicio example-vault que conecta por puerto 8200, pero sin rutas o ingress asociados.
* Replica set example-vault-agent-injector.
* Stateful set example-vault.

## Documentación

Para instalar Vault con Helm según OpenShift

* <https://docs.openshift.com/container-platform/4.10/applications/working_with_helm_charts/installing-helm.html>
* <https://docs.openshift.com/container-platform/4.10/applications/working_with_helm_charts/configuring-custom-helm-chart-repositories.html>

Complementario

* <https://developer.hashicorp.com/vault/docs/platform/k8s/helm/openshift>
* <https://cloud.redhat.com/blog/integrating-hashicorp-vault-in-openshift-4>

## Installing a Helm chart on an OpenShift Container Platform cluster

Fuente: <https://docs.openshift.com/container-platform/4.10/applications/working_with_helm_charts/configuring-custom-helm-chart-repositories.html>

*Prerequisites*

*You have a running OpenShift Container Platform cluster and you have logged into it.*

*You have installed Helm.*

Procedimiento

*Create a new project:*

$ oc new-project vault

Now using project "vault" on server "https://api.k8spro.nextret.net:6443".

You can add applications to this project with the 'new-app' command. For example, try:

oc new-app rails-postgresql-example

to build a new example application in Ruby. Or use kubectl to deploy a simple Kubernetes application:

kubectl create deployment hello-node --image=k8s.gcr.io/serve\_hostname

*Add a repository of Helm charts to your local Helm client:*

$ helm repo add openshift-helm-charts https://charts.openshift.io/

"openshift-helm-charts" has been added to your repositories

*Update the repository:*

$ helm repo update

Hang tight while we grab the latest from your chart repositories...

...Successfully got an update from the "openshift-helm-charts" chart repository

Update Complete. Happy Helming!

*Install an example HashiCorp Vault:*

$ helm install example-vault openshift-helm-charts/hashicorp-vault

NAME: example-vault

LAST DEPLOYED: Tue Jun 13 12:53:24 2023

NAMESPACE: vault

STATUS: deployed

REVISION: 1

NOTES:

Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using

Vault with Kubernetes available here:

https://www.vaultproject.io/docs/

Your release is named example-vault. To learn more about the release, try:

$ helm status example-vault

$ helm get manifest example-vault

Con helm status example-vault se obtiene la misma salida anterior.

Lista de instalaciones de Helm.

> helm list

NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION

example-vault vault 1 2023-06-13 12:53:24.3780837 +0200 CEST deployed vault-0.24.0 1.13.1

Manifiestos K8s generados por la instalación.

> helm get manifest example-vault

---

# Source: vault/templates/injector-network-policy.yaml

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: example-vault-agent-injector

(...)

# Instalación de Vault

La instalación de Vault se realizón en dos intentos:

* El primero según la documentación oficial de OpenShift que tenía algunas restricciones que nos hace descartar el método. Ver detalles en [Instalación de Vault según la documentación de OpenShift](" \l "11_vault_fail_md).
* El segundo como parte de un tutorial que satisface lo que se considera necesario.

Esta segunda referencia es muy similar pero  
solventa los problemas vistos en la primera opción.  
<https://medium.com/hybrid-cloud-engineering/vault-integration-into-openshift-container-platform-b57c175a79da>

Cuidado: Esta referencia está bien para instalar Vault, pero luego usa el método más complejo de inyección al contenedor. El objetivo es actualizar un secreto, simplemente.

## Instalación con UI e inicialización

### Crear un namespace para todo lo Vault

*First, create a namespace in OCP called vault-infra using the following command:*

oc create namespace vault-infra

namespace/vault-infra created

*This namespace will be the home for Vault in your OCP environment. Confirm that the new namespace is created with the following command:*

oc get namespaces | grep vault-infra

vault-infra Active 89s

### Clonar localmente el Helm chart de Vault

*Clone the vault-helm GitHub repository into your local filesystem (using the command below). This step makes it much easier to keep track of the configuration changes you make to the Vault helm chart before you install it.*

git clone https://github.com/hashicorp/vault-helm.git

Cloning into 'vault-helm'...

remote: Enumerating objects: 4008, done.

remote: Counting objects: 100% (1843/1843), done.

remote: Compressing objects: 100% (300/300), done.

remote: Total 4008 (delta 1710), reused 1588 (delta 1537), pack-reused 2165

Receiving objects: 100% (4008/4008), 1.05 MiB | 4.38 MiB/s, done.

Resolving deltas: 100% (3010/3010), done.

### Ajustar el clon local del Helm chart de Vault

*The most important piece is the values.yaml file. The Vault helm chart will use this file to customize the Vault instance to your specifications. There are a large number of different configuration options ranging from high availability (HA), audit logging, probes, and more. For the purposes of this article, I will give the configuration for a quick and easy install.*

*In the values.yaml change the following keys:*

No es casual que se deba activar la opción de OpenShift, donde desplegamos,  
y las de ruta y UI que eran las que se echaban en falta.

global:

openshift: true

server:

route: true

ui:

enabled: true

### Definir un nombre para la ruta de acceso

Atención! En values.yaml hay varias entradas que definen chart-example.local como nombre de la ruta/ingress para acceder a la consola de Vault.

El guión en el nombre no funciona con el etc/hosts de Windows. Cambiar chart-example.local por un nombre que se pueda resolver por este fichero,  
o bien si funciona el DNS con el sufijo del cluster poner un nombre que encaje,  
por ejemplo: vaultinfra.apps.k8spro.nextret.net.

Nótese como se define como ruta, ingress inactivo, cuando se trata de deplegar a OpenShift.

ingress:

enabled: false

hosts:

- host: vaultinfra.apps.k8spro.nextret.net

# OpenShift only - create a route to expose the service

# By default the created route will be of type passthrough

route:

enabled: true

host: vaultinfra.apps.k8spro.nextret.net

### Instalar Vault a partir del Helm Chart moldificado

*Change directory into the vault-helm sub-directory and run the following command:*

helm install vault . -n vault-infra

La salida es similar a la opción probada en primer lugar y que se descartó.

Probar helm get manifest vault.

NAME: vault

LAST DEPLOYED: Tue Jun 13 13:24:57 2023

NAMESPACE: vault-infra

STATUS: deployed

REVISION: 1

NOTES:

Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using

Vault with Kubernetes available here:

https://www.vaultproject.io/docs/

Your release is named vault. To learn more about the release, try:

$ helm status vault

$ helm get manifest vault

## Tareas post instalación

### Unseal del Vault

Tal cual acaba de arrancar, el pod vault-0 del statefulset no está ready por estar pendiente del unsealing.

*You can then run the following command to check the status of your Vault deployment:*

oc get pods -n vault-infra

NAME READY STATUS RESTARTS AGE

vault-0 0/1 Running 0 5m9s

vault-agent-injector-76f7d7d848-tqf5s 1/1 Running 0 5m9s

*What you will see is that the vault-0 pod is stuck in the Ready: 0/1 state. This is because Vault needs to be initialized and unsealed before it can be used.*

*To see a more descriptive error message, navigate into the OpenShift console UI and navigate to Pods > vault-0 > Logs.*

En el log se ve este fragmento, cuyas tres últimas líneas indican que se debe proceder al usnealing para que se inicialice Vault.

==> Vault server configuration:

Api Address: http://172.17.5.214:8200

Cgo: disabled

Cluster Address: https://vault-0.vault-internal:8201

Environment Variables: GODEBUG, HOME, HOSTNAME, HOST\_IP, KUBERNETES\_PORT, KUBERNETES\_PORT\_443\_TCP, KUBERNETES\_PORT\_443\_TCP\_ADDR, KUBERNETES\_PORT\_443\_TCP\_PORT, KUBERNETES\_PORT\_443\_TCP\_PROTO, KUBERNETES\_SERVICE\_HOST, KUBERNETES\_SERVICE\_PORT, KUBERNETES\_SERVICE\_PORT\_HTTPS, NAME, NSS\_SDB\_USE\_CACHE, PATH, POD\_IP, PWD, SHLVL, SKIP\_CHOWN, SKIP\_SETCAP, TERM, VAULT\_ADDR, VAULT\_AGENT\_INJECTOR\_SVC\_PORT, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP\_ADDR, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP\_PORT, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP\_PROTO, VAULT\_AGENT\_INJECTOR\_SVC\_SERVICE\_HOST, VAULT\_AGENT\_INJECTOR\_SVC\_SERVICE\_PORT, VAULT\_AGENT\_INJECTOR\_SVC\_SERVICE\_PORT\_HTTPS, VAULT\_API\_ADDR, VAULT\_CLUSTER\_ADDR, VAULT\_K8S\_NAMESPACE, VAULT\_K8S\_POD\_NAME, VAULT\_PORT, VAULT\_PORT\_8200\_TCP, VAULT\_PORT\_8200\_TCP\_ADDR, VAULT\_PORT\_8200\_TCP\_PORT, VAULT\_PORT\_8200\_TCP\_PROTO, VAULT\_PORT\_8201\_TCP, VAULT\_PORT\_8201\_TCP\_ADDR, VAULT\_PORT\_8201\_TCP\_PORT, VAULT\_PORT\_8201\_TCP\_PROTO, VAULT\_SERVICE\_HOST, VAULT\_SERVICE\_PORT...

Go Version: go1.20.1

Listener 1: tcp (addr: "[::]:8200", cluster address: "[::]:8201", max\_request\_duration: "1m30s", max\_request\_size: "33554432", tls: "disabled")

Log Level:

Mlock: supported: true, enabled: false

Recovery Mode: false

Storage: file

Version: Vault v1.13.1, built 2023-03-23T12:51:35Z

Version Sha: 4472e4a3fbcc984b7e3dc48f5a8283f3efe6f282

==> Vault server started! Log data will stream in below:

2023-06-13T11:25:16.786Z [INFO] proxy environment: http\_proxy="" https\_proxy="" no\_proxy=""

2023-06-13T11:25:16.787Z [INFO] core: Initializing version history cache for core

2023-06-13T11:25:23.421Z [INFO] core: security barrier not initialized

2023-06-13T11:25:23.421Z [INFO] core: seal configuration missing, not initialized

2023-06-13T11:25:28.446Z [INFO] core: security barrier not initialized

2023-06-13T11:25:28.447Z [INFO] core: seal configuration missing, not initialized

De paso, al principio del log salen las variables de entorno definidas, algunas de las cuales usaremos luego.

Environment Variables: GODEBUG, HOME, HOSTNAME, HOST\_IP, KUBERNETES\_PORT, KUBERNETES\_PORT\_443\_TCP, KUBERNETES\_PORT\_443\_TCP\_ADDR, KUBERNETES\_PORT\_443\_TCP\_PORT, KUBERNETES\_PORT\_443\_TCP\_PROTO, KUBERNETES\_SERVICE\_HOST, KUBERNETES\_SERVICE\_PORT, KUBERNETES\_SERVICE\_PORT\_HTTPS, NAME, NSS\_SDB\_USE\_CACHE, PATH, POD\_IP, PWD, SHLVL, SKIP\_CHOWN, SKIP\_SETCAP, TERM, VAULT\_ADDR, VAULT\_AGENT\_INJECTOR\_SVC\_PORT, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP\_ADDR, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP\_PORT, VAULT\_AGENT\_INJECTOR\_SVC\_PORT\_443\_TCP\_PROTO, VAULT\_AGENT\_INJECTOR\_SVC\_SERVICE\_HOST, VAULT\_AGENT\_INJECTOR\_SVC\_SERVICE\_PORT, VAULT\_AGENT\_INJECTOR\_SVC\_SERVICE\_PORT\_HTTPS, VAULT\_API\_ADDR, VAULT\_CLUSTER\_ADDR, VAULT\_K8S\_NAMESPACE, VAULT\_K8S\_POD\_NAME, VAULT\_PORT, VAULT\_PORT\_8200\_TCP, VAULT\_PORT\_8200\_TCP\_ADDR, VAULT\_PORT\_8200\_TCP\_PORT, VAULT\_PORT\_8200\_TCP\_PROTO, VAULT\_PORT\_8201\_TCP, VAULT\_PORT\_8201\_TCP\_ADDR, VAULT\_PORT\_8201\_TCP\_PORT, VAULT\_PORT\_8201\_TCP\_PROTO, VAULT\_SERVICE\_HOST, VAULT\_SERVICE\_PORT...

*Initialize and Unseal Vault*

*To initialize and unseal Vault, use the Vault CLI tool. The Vault CLI comes preinstalled in the vault container within the vault-0 pod, so I will use the Terminal tab of the container for all following commands. Alternatively, you can exec into the pod or install the Vault CLI tool locally and use your own terminal to run Vault commands.*

*To first initialize the Vault, run the following command:*

Desde el terminal del pod vault-0 (ir al statefulset y buscar sus pods).  
Tendremos un primer error que corregiremos.

# vault version

Vault v1.13.1 (4472e4a3fbcc984b7e3dc48f5a8283f3efe6f282), built 2023-03-23T12:51:35Z

# vault operator init

Error initializing: Error making API request.

URL: PUT http://127.0.0.1:8200/v1/sys/init

Code: 400. Errors:

\* failed to initialize barrier: failed to persist keyring: mkdir /vault/data/core: permission denied

Corregimos el error dando permisos al directorio para una install de pruebas. Luego habrá que probar cual es la mejor solución.

chmod a+w /vault/data

Corregido el error y con el comando de inicialización salen

* Las 5 claves de unseal, con 3 de las cuales podremos hacer unseal ya que son el quorum necesario.
* El root token que servirá para hacer login en un Vault unsealed.

vault operator init

Unseal Key 1: TBf03DweOoMQ3A3EvxqS/wX4kZb1JV3IQOQAkNK79v7c

Unseal Key 2: 7lSvenOE5mDq3O1xJlBqs5q4FrHJDpxT9Uri3m7FW+lH

Unseal Key 3: f6HwDGcyMcvbXKOJMbaPQS9TjuxiOySeR/YIHSGxZLli

Unseal Key 4: niF2XCa0pBCC5Hvxq/kH+SxximC6Hjm8Sfdkq0NQ+8kM

Unseal Key 5: YqBZ0yTFUl4YYFQmFQS91+Nlznyp/YN7aWGVlaGQ9abv

Initial Root Token: hvs.MbBT9J9ZXyGvlJCjV8CG1IjD

Vault initialized with 5 key shares and a key threshold of 3. Please securely

distribute the key shares printed above. When the Vault is re-sealed,

restarted, or stopped, you must supply at least 3 of these keys to unseal it

before it can start servicing requests.

Vault does not store the generated root key. Without at least 3 keys to

reconstruct the root key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of

existing unseal keys shares. See "vault operator rekey" for more information.

*As stated in the text of the image above, Vault needs 3/5 of the unseal keys to unseal the Vault. It is very important to remember these keys! In addition, it is important to keep track of the initial root token, because this will allow login and configuration of your Vault resources. To unseal the Vault, run the following command with 3 of your unseal keys:*

*The most important pieces are the Sealed=true and Unseal Progress key-value pairs.*

Basta con ir ejecutando el comando vault operator unseal con hasta tres de las claves.

vault operator unseal TBf03DweOoMQ3A3EvxqS/wX4kZb1JV3IQOQAkNK79v7c

Key Value

--- -----

Seal Type shamir

Initialized true

--> Sealed true

Total Shares 5

Threshold 3

--> Unseal Progress 1/3

Unseal Nonce 9ecf62da-64e9-47d4-8967-a9b87a1e1c76

Version 1.13.1

Build Date 2023-03-23T12:51:35Z

Storage Type file

HA Enabled false

vault operator unseal 7lSvenOE5mDq3O1xJlBqs5q4FrHJDpxT9Uri3m7FW+lH

Sealed true

Unseal Progress 2/3

vault operator unseal f6HwDGcyMcvbXKOJMbaPQS9TjuxiOySeR/YIHSGxZLli

Sealed false

*Success, Vault is now initialized and unsealed! I checked my running pods and found that the vault-0 pod changed to a Ready: 1/1 state.*

Una vez inicializado, el pod se manifiesta como arrancado.

>oc get pods -n vault-infra

NAME READY STATUS RESTARTS AGE

vault-0 1/1 Running 0 26m

vault-agent-injector-76f7d7d848-tqf5s 1/1 Running 0 26m

*Before running the Vault commands in the following sections, you need to login to Vault using the root token acquired in the last section. The root token will also be used later on to login to the Vault UI. To do so, run the following command:*

Como se decía, login con el root para poder ejecutar comandos de Vault en el pod vault-0.

vault login hvs.MbBT9J9ZXyGvlJCjV8CG1IjD

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 hvs.LRpjkv1Kg0o9Gmj7N9bgkRDz

token\_accessor nw6T3xFMklkwu0Qht1JL13es

token\_duration ∞

token\_renewable false

token\_policies ["root"]

identity\_policies []

policies ["root"]

### Playground: Crear secretos

*Now that all the permissions are set up, it is time to create a secret to store into Vault. The example application in the later section of this article expects a database username and password stored in the ceh/database/credentials Vault path. To create the database secret, first create the ceh path with the following command:*

Un pequeño ejemplo de creación de secretos en la fuente de referencia.

Primero es necesario habilitar el path raíz ceh del path del secreto, y asociarle un tipo kv-v2 de motor de secretos.

vault secrets enable -path=ceh kv-v2

Success! Enabled the kv-v2 secrets engine at: ceh/

*This creates the ceh path with the kv-v2 secrets engine. You can find more information about Vault secrets engines here. Create a username and password entry with the following command:*

Ya se puede crear un secreto en el path ceh/database/credentials bajo el path autorizado anteriormente.  
Nótese que aunque el path que veremos en la GUI es el que pasamos en el comando,  
el que usaremos más adelante en las configuraciones  
es el que aparece en la respuesta bajo Secret Path  
y en el que se intercala un /data tras el directorio raíz habilitado.

vault kv put ceh/database/credentials \

username="db-username" \

password="db-password"

======== Secret Path ========

ceh/data/database/credentials

======= Metadata =======

Key Value

--- -----

created\_time 2023-06-13T11:55:01.028736078Z

custom\_metadata <nil>

deletion\_time n/a

destroyed false

version 1

Este secreto se puede ver ya en la GUI en <https://vaultinfra.apps.k8spro.nextret.net/ui/vault/secrets/ceh/show/database/credentials>.  
Pero no, ver siguiente apartado.

Nota: De nuevo ver como se separa la raíz del resto con un /show/ intermedio.

### Configurar TLS de la ruta

Al acceder a <https://vaultinfra.apps.k8spro.nextret.net/ui/vault/secrets/ceh/show/database/credentials>  
el navegador recibe una respuesta incorrecta debido a que el tráfico https se pasa tal cual a Vault en :8200.

Basta cambiar la ruta para tener tls.termination: edge. El tls.termination: passthrough anterior viene del Helm que ya se había modificado para el nombre del host.

spec:

host: vaultinfra.apps.k8spro.nextret.net

to:

kind: Service

name: vault

weight: 100

port:

targetPort: 8200

tls:

termination: edge

Sobre las terminaciones de TLS:

*Routes can be either secured or unsecured. Secure routes provide the ability to use several types of TLS termination to serve certificates to the client. Unsecured routes are the simplest to configure, because they require no key or certificates, but secured routes encrypt traffic to and from the pods. A secured route specifies the TLS termination of the route. The available types of termination are listed below:*

* *Edge Termination With edge termination, TLS termination occurs at the router, before the traffic gets routed to the pods. TLS certificates are served by the router, so they must be configured into the route, otherwise the router’s default certificate is used for TLS termination. Because TLS is terminated at the router, connections from the router to the endpoints over the internal network are not encrypted.*
* *Pass-through Termination With pass-through termination, encrypted traffic is sent straight to the destination pod without the router providing TLS termination. No key or certificate is required. The destination pod is responsible for serving certificates for the traffic at the endpoint.*
* *Re-encryption Termination Re-encryption is a variation on edge termination, where the router terminates TLS with a certificate, then re-encrypts its connection to the endpoint, which might have a different certificate. Therefore the full path of the connection is encrypted, even over the internal network.*

Y ya en la pantalla, entramos con método Token indicando el valor del root token hvs.MbBT9J9ZXyGvlJCjV8CG1IjD  
y podemos navegar al secreto antes creado <https://vaultinfra.apps.k8spro.nextret.net/ui/vault/secrets/ceh/show/database/credentials>.

### Autenticación de Kubernetes

*Configuring Kubernetes Authentication*

*Before the Vault secrets can be injected into a pod, the pod needs to be able to authenticate itself to the Vault service. Vault supports multiple authentication methods which can be found here. For this article I used the Kubernetes auth method because of how easy it is to set up. To first enable the auth method endpoint, run the following command:*

Este método de autenticación no está previsto en esta prueba de concepto.

# Instalación de ArgoCD

ArgoCD, al igual que Vault, es una de las precondiciones necesarias para poder usar el ArgoCD Vault Plugin (AVP).

Para poder configurar ArgoCD a nuestro antojo (Vault lo requiere, o Deus Vault si se prefiere)  
es mejor no instalar el operador de ArgoCD para Openshift  
sino instalar el **operador OpenShift GitOps** que es el propio de OpenShift.

## Instalación de operador OpenShift GitOps

Seguir lo aquí indicado, instalando para todos los namespaces del cluster.

* <https://github.com/redhat-developer/openshift-gitops-getting-started>

*Log into OpenShift Web Console as a cluster admin and navigate to the Administrator perspective and then Operators → OperatorHub.*

*In the OperatorHub, search for OpenShift GitOps and follow the operator install flow to install it.*

## Instancia personalizada de ArgoCD

En próximos intentos descubriremos que es necesario  
hacer un despliegue de ArgoCD personalizado (con el Vault Plugin)  
ya que no se puede modificar el gestionado por openshift-gitops  
(un operador muy temperamental sobre lo que él gestiona).

Para empezar a probar el despliegue de una instancia personalizada seguiremos el artículo:  
<https://github.com/redhat-developer/openshift-gitops-getting-started#additional-argo-cd-instances>

*Additional Argo CD instances*

*Although OpenShift GitOps by default installs an Argo CD instance for the cluster, there are use-cases where different application teams might need their own Argo CD instance confined to their own namespaces and applications. Therefore, OpenShift GitOps support creating additional Argo CD instances declaratively through creating ArgoCD resources.*

*In the OpenShift Web Console, create a project called myargocd and then click on the plus sign in the top navigation bar. Then, paste the following in the YAML editor, and click on Create afterwards:*

En la consola crear el namespace myargocd y aplicar este manifiesto desde el icono del símbolo **+**

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

metadata:

name: myargocd

spec:

server:

route:

enabled: true

*Alternatively, you can run the following CLI commands:*

Se desconoce de donde sale este argo/argocd.yaml, o sea que mejor aplicar el YAML anterior.

oc new-project myargocd

oc create -f argo/argocd.yaml

### Acceso a los servidores ArgoCD

*Click on the Topology to view the Argo CD instance deployed in your namespace.*

*Click on the Argo CD URL to open the Argo CD dashboard.*

El pod myargocd-server es el que tiene la flecha con el hover Open URL.

*As described previously, Argo CD upon installation generates an initial admin password which is stored in a Kubernetes secret called [argocd-name]-cluster. Run the following command to decrypt the admin password and log into Argo CD dashboard:*

Método que proponen:

oc get secret myargocd-cluster -n myargocd -ojsonpath='{.data.admin\.password}' | base64 -d

5Jmx...Tq8

También se puede usar el método tradicional que no requiere entrar a un bash para tener base64:

oc extract secret/myargocd-cluster -n myargocd --to=-

# admin.password

5Jmx...Tq8

Para el operador openshift-gitops en otro namespace

oc extract secret/openshift-gitops-cluster -n openshift-gitops --to=-

Como resultado, quedan dos servidores ArgoCD accesibles, el por defecto y el que se acaba de instalar, respectivamente:

* <https://openshift-gitops-server-openshift-gitops.apps.k8spro.nextret.net/applications>
* <https://myargocd-server-myargocd.apps.k8spro.nextret.net/applications>

NB: En la vista de topología está en enlace a cada argo server, pero al final la URL no deja de ser:

https://${namespace}-server-${namespace}.apps.${cluster}

## Utilidades

### Instalación de ArgoCD línea de comandos

El cliente de ArgoCD de línea de comandos puede ser de utilidad a futuro.

*The argocd CLI makes it easier to work with Argo CD. Through it, you can manage Argo CD projects, applications, cluster credentials, and more.*

*To install the argocd CLI, follow these steps:*

* Download the latest Argo CD binary file from <https://github.com/argoproj/argo-cd/releases/latest>.
* If you are using Linux, download the CLI and add it to your path:

$ sudo curl -sSL -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64

$ sudo chmod +x /usr/local/bin/argocd

*If everything went well, you will see the following output by running the argocd version command. Ignore the error message you see in the last line; it is an expected message as we haven’t logged in to any OpenShift cluster yet:*

argocd version

argocd: v2.2.1+122ecef

BuildDate: 2021-12-17T01:31:40Z

GitCommit: 122ecefc3abfe8b691a08d9f3cecf9a170cc8c37

GitTreeState: clean

GoVersion: go1.16.11

Compiler: gc

Platform: linux/amd64

FATA[0000] Argo CD server address unspecified

### API de ArgoCD

La API de ArgoCD puede ser útil a futuro para automatizar operaciones administrativas de recursos de ArgoCD.

* <https://myargocd-server-myargocd.apps.k8spro.nextret.net/swagger-ui>
* <https://argo-cd.readthedocs.io/en/stable/developer-guide/api-docs/>

Ejemplo de login via API. -kpara evitar el error de TLS.

ARGOCD\_SERVER=https://myargocd-server-myargocd.apps.k8spro.nextret.net

curl -k $ARGOCD\_SERVER/api/v1/session -d $'{"username":"admin","password":"5Jm...Tq8"}'

## ArgoCD en cualquier namespace

La instalación personalizada es capaz de desplegar solo en su propio namespace.

Para desplegar en un conjunto de namespaces (restringido por seguridad) se debería seguir esta guía.

* <https://argo-cd.readthedocs.io/en/stable/operator-manual/app-any-namespace/>

No obstante, en otro punto de este documento se detalla como crear el namespace ya manejado por el namespace de la instancia de ArgoCD, sin más complicación.

# ArgoCD Vault Plugin (AVP)

Esta página contiene documentación y presentación de ArgoCD Vault Plugin (AVP).

Para el proceso de instalación y configuración de AVP en una prueba de concepto, consultar:

* [Instalación de ArgoCD Vault Plugin (AVP)](" \l "21_avp_install_md).
* [Autenticación y autorización por approle en Vault](#22_avp_vault_approle_md).
* [Despliegue de aplicaciones en ArgoCD con Vault approle](#23_avp_argocd_approle_md).

## Documentación de AVP

Como documentación de referencia, la página de proyecto y la de readthedocs.io.

* <https://github.com/argoproj-labs/argocd-vault-plugin>
* <https://argocd-vault-plugin.readthedocs.io/>

Un par de páginas de utilidad: los backends compatibles para almacenar secretos, y una lista de releases para estar al día de las disponibles.

* <https://argocd-vault-plugin.readthedocs.io/en/stable/backends/>
* <https://github.com/argoproj-labs/argocd-vault-plugin/releases>

Esta guía propone un par de variantes que se pueden usar algún día: Crear una imagen de ArgoCD con el plugin instalado, y configurar la autenticación en Vault usando el service account que ejecuta en Kubernetes.

* <https://cloud.redhat.com/blog/how-to-use-hashicorp-vault-and-argo-cd-for-gitops-on-openshift>

## Presentación de AVP

El resto de esta página contiene la parte descriptiva del artículo.  
<https://itnext.io/argocd-secret-management-with-argocd-vault-plugin-539f104aff05>.

*GitOps has quickly become one of the hotter topics within the realm of DevOps. GitOps was introduced by Weaveworks in 2017 and has been trending upward ever since. I will not go into why to use GitOps in this article but you can read more about it here. One of the questions that always comes up when discussing GitOps is Secret Management. Every single talk, presentation or demo involving GitOps always has someone bringing up the question, “How do you handle secrets with GitOps?”, and that is a very good question.*

*As our team was evaluating moving to GitOps, we landed on using ArgoCD as our GitOps solution. ArgoCD provided a stable tool that could handle deploying hundreds of microservices across many different Kubernetes Clusters in a fast and reliable way.*

*Once we chose our GitOps tool, it was time to figure out what to do about the Secrets problem. At the same time, we were starting to migrate our Secrets to HashiCorp Vault, so we knew we would need something that could bridge the gap between ArgoCD and Vault. We looked around at some existing tools and one of the issues we found were that the potential solutions to this problem had a very high barrier to entry. Whether that was having to manually encrypt secrets or deploying Operators to do some of the work, none of these solutions fit well with what my team was trying to do.*

*So, we decided to build our own tool called argocd-vault-plugin.*

*What is the argocd-vault-plugin?*

*The argocd-vault-plugin is a custom ArgoCD plugin for retrieving secrets from HashiCorp Vault and injecting them into Kubernetes YAML files. Within ArgoCD, there is a way to integrate custom plugins if you need something outside of the supported tools that are built-in and we wanted to take advantage of this pattern. One of the ideas behind this plugin was to write it in a way that did not require an Operator or Custom Resource. This allows us to be able to parameterize any Kubernetes resources, even Custom Resources, not just Secrets.*

*The plugin works by first retrieving values from Vault based on a path that can be specified as an Environment Variable or an Annotation inside of the YAML file and then injects the values into a templated out yaml, that uses <> as the template markers. For example:*

kind: Secret

apiVersion: v1

metadata:

name: example-secret

annotations:

avp.kubernetes.io/path: "path/to/secret"

type: Opaque

stringData:

password: <password-vault-key>

*In the yaml above, you see that we have a normal Kubernetes Secret definition. However, something is not quite normal. Under data, we have a secret named password but the value is <password-vault-key> this is where the plugin will inject a value if it finds the password-vault-key key in Vault.*

*So when the plugin runs, it will take the avp.kubernetes.io/pathannotation from the yaml and use that to look for the secrets we want to inject into this Kubernetes Secret. So we take the avp.kubernetes.io/path annotation and combine it with the value inside the <> symbols. So the final path in this scenario would be path/to/secret with the Vault key password-vault-key. So when the plugin finishes running, we would have yaml that looks like this:*

kind: Secret

apiVersion: v1

metadata:

name: example-secret

annotations:

avp.kubernetes.io/path: "path/to/secret"

type: Opaque

stringData:

password: some-password # From the key password-vault-key in Vault

*You can view all the supported backends and authentication types here,* [*https://argocd-vault-plugin.readthedocs.io/en/stable/backends/*](https://argocd-vault-plugin.readthedocs.io/en/stable/backends/)*.*

# Instalación de ArgoCD Vault Plugin (AVP)

Instalar AVP se resume a que en uno de los servidores de ArgoCD se disponga de  
un binario ejecutable capaz de hacer cambios en los manifiestos que maneja ArgoCD. Este binario debe:

* Estar registrado como plugin para que se pueda seleccionar durante la creación de una aplicación ArgoCD.
* Ser capaz de recibir configuración de como acceder a un repositorio de secretos externo, lo que se hará a través de variables de entorno.
* Con la configuración dada, acceder al repositorio de secretos y con lo obtenido realizar cambios en los manifiestos, poniendo en ellos información sensible que no debiera estar en ningún repositorio de código abierto a cualquiera.  
  Recordatorio: Los manifiestos para poder ser enriquecidos con secretos por ArgoCD deben estar anotados, la anotación debe dar un path o información de como llegar al secreto deseado en el sistema externo configurado, y tener unas marcas de sustitución con el nombre atributo del secreto rodeado por < y >.

## Retos, opciones y alternativas

**Forma de transferir los secretos**

En esta prueba se hace una instalación del tipo que ArgoCD despliega secretos de K8s que contienen los secretos de Valt visibles para administradores de cluster y namespace. Es un riesgo asumido y asumible si las políticas de seguridad en efecto lo permiten.

**Tipo de instalación del plugin**

Existe la alternativa de usar contenedores inyectores, que durante el ciclo de vida del contenedor principal inyectan en la memoria de este el secreto acabado de leer de Vault.

* <https://developer.hashicorp.com/vault/docs/platform/k8s/injector>

La instalación tal como se propone plantea varias opciones:

* La forma de instalar el AVP usada en esta prueba de concepto es la de  
  los llamados initContainer que arrancan con el contenedor compartiendo espacio de disco.  
  El initContainer es quien se encarga de obtener el plugin y dejarlo en el contenedor principal antes de salir de escena al acabar la inicialización.
* Hay alternativas como crear imágenes del servidor ArgoCD con el plugin ya instalado.

**Modificación de componentes del operador de ArgoCD**

Las configuraciones requeridas hacen necesario reconfigurar alguno de los servidores de ArgoCD.

Se ha observado que el operador gitops-operator o "ArgoCD de Red Hat" no siempre permite estas reconfiguraciones, y se ha optado por una instancia personalizada.

**TTL de los secret id en Vault approle**

En este modo de autenticación y autorización en Vault, existe un id de rol inmutable (asimilable al "nombre de usuario" del rol) y un secret id que hace las funciones de password del rol.  
El id de secreto tiene una validez temporal de máximo 720 horas o 30 días. Suficiente para una prueba de concepto, pero no para explotación.

Esta limitación es buscada para forzar una política de renovación de secretos, entendida como buena práctica.

## Instalación ArgoCD Vault Plugin (AVP)

Seguiremos la sección **How to Install and use the Plugin** de <https://itnext.io/argocd-secret-management-with-argocd-vault-plugin-539f104aff05>  
hasta llegar al punto en que se empiezan a configurar approles. En ese momento seguiremos en  
[Autenticación y autorización por approle en Vault](#22_avp_vault_approle_md) y  
[Despliegue de aplicaciones en ArgoCD con Vault approle](#23_avp_argocd_approle_md) donde se explica en detalle al margen de la instalación.

### Make Plugin Available to be used by ArgoCD

*In order to use the plugin, we first need to download the plugin and then register the plugin with ArgoCD to use as a custom plugin. There are a couple different methods for doing this that you can read about here but for this example we are going to use the Volume Mount strategy. In order to do this, we will have to customize the argocd-repo-server deployment:*

Se intenta inyectar un initContainer en el repo server para que al arrancar descargue el plugin y lo copie donde toca.

**Cómo no hacerlo**

O también "con openshift-gitops hemos topado":

* Si se edita el deployment del repo server mientras este ejecuta:
* Error "Forbidden: pod updates may not add or remove containers" for field "spec.initContainers".
* Si se para y se edita, salta este warning y no se guarda. Soltar el warning forma parte de lo correcto en OpenShift  
  <http://openshift.github.io/openshift-origin-design/designs/administrator/olm/warn-managed-resource/>.
* This resource is managed by ArgoCD
* ACD
* openshift-gitops
* and any modifications may be overwritten. Edit the managing resource to preserve changes.
* http://openshift.github.io/openshift-origin-design/designs/administrator/olm/warn-managed-resource/

**Qué hay que montar**

Estos son los cambios necesarios para poder inyectar el plugin AVP con  
el initContainer download-tools en el repo server,  
con el que comparte un volumeMount llamado /custom-tools.  
El volumen compartido se monta en un directorio de trabajo del initContainer,  
pero para el repo server es el directorio de binarios en el path del sistema.

containers:

- name: argocd-repo-server

volumeMounts:

- name: custom-tools

mountPath: /usr/local/bin/argocd-vault-plugin

subPath: argocd-vault-plugin

volumes:

- name: custom-tools

emptyDir: {}

initContainers:

- name: download-tools

image: alpine:3.8

command: [sh, -c]

args:

- wget -O argocd-vault-plugin

https://github.com/argoproj-labs/argocd-vault-plugin/releases/download/v1.6.0/argocd-vault-plugin\_1.6.0\_linux\_amd64

chmod +x argocd-vault-plugin &&\

mv argocd-vault-plugin /custom-tools/

volumeMounts:

- mountPath: /custom-tools

name: custom-tools

### Inserción del plugin

En el namespace donde esté la instancia personalizada de ArgoCD (myargocd en esta prueba),  
ir al despliegue del argocd-repo-server.  
En su ficha aparece debajo del nombre un enlace managed by.  
Acceder a ese recurso de tipo ArgoCD y editar el YAML  
cambiando el repo {} por esta estructura con  
volumes, volumeMounts e initContainers:

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

spec:

repo:

volumes:

- emptyDir: {}

name: custom-tools

volumeMounts:

- mountPath: /usr/local/bin/argocd-vault-plugin

name: custom-tools

subPath: argocd-vault-plugin

initContainers:

- command:

- sh

- '-c'

args:

- >-

cd /tmp

wget -O argocd-vault-plugin https://github.com/argoproj-labs/argocd-vault-plugin/releases/download/v1.6.0/argocd-vault-plugin\_1.6.0\_linux\_amd64

chmod +x argocd-vault-plugin && mv argocd-vault-plugin /custom-tools/

image: 'alpine:3.8'

name: download-tools

volumeMounts:

- mountPath: /custom-tools

name: custom-tools

Es interesante ver como en el arranque del initContainer se va  
al directorio /tmp para hacer una descarga a la que luego se le dan permisos de ejecución y luego se envía al directorio compartido con el repo server.

Se propone que la descarga del plugin sea de un repositorio controlado y no directamente de github.

Para depurar el arranque del initContainer con oc logs  
hay que usar la opción -c download-tools después del id del pod, según  
<https://kubernetes.io/docs/tasks/debug/debug-application/debug-init-containers/>

oc logs myargocd-repo-server-74cbcc69b7-wzfkw -c download-tools

Connecting to github.com (140.82.121.3:443)

Connecting to objects.githubusercontent.com (185.199.108.133:443)

argocd-vault-plugin 42% |\*\*\*\*\*\*\*\*\*\*\*\*\* | 17335k 0:00:01 ETA

argocd-vault-plugin 74% |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | 30119k 0:00:00 ETA

argocd-vault-plugin 92% |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | 37607k 0:00:00 ETA

argocd-vault-plugin 100% |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| 40672k 0:00:00 ETA

En este punto cuando ya ha arrancado el contenedor principal con el plugin copiado y ejecutable,  
se puede entrar por terminal y validar que ejecute.

argocd-vault-plugin

This is a plugin to replace <placeholders> with Vault secrets

Usage:

argocd-vault-plugin [flags]

argocd-vault-plugin [command]

Available Commands:

completion generate the autocompletion script for the specified shell

generate Generate manifests from templates with Vault values

help Help about any command

version Print argocd-vault-plugin version information

Flags:

-h, --help help for argocd-vault-plugin

Use "argocd-vault-plugin [command] --help" for more information about a command.

argocd-vault-plugin version

argocd-vault-plugin v1.6.0 (947668d260d7e630b3dbc7d9dadfc4ed0650ccd3) BuildDate: 2021-12-01T21:37:22Z

`

Esta ejecución a demanda se puede hacer incluso en otro Linux incluído WSL-2 en Windows, lo que resultó esclarecedor para trazar algún error de configuración (como se verá, pasando variables de entorno).

### Registrar el plugin en ArgoCD

Por plugin disponible (available) se entiende que el repo server ya lo puede ejecutar, aunque de momento no es más que un binario en el path de ejecutables.

*Once the plugin has been made available,  
the next step is to register the plugin with ArgoCD itself.  
This is a pretty straight forward step.  
There is a configMap called argocd-cm. All that is required to to go to that configMap and add:*

En el configMap argocd-cm del namespace myargocd se efectúa este cambio.  
Básicamente se le dice a ArgoCD que dispone de este plugin que durante la fase generate de ArgoCD deberá ser llamado con los parámetros para generar en el directorio actual.  
A los parámetros args del plugin se le pueden añadir otros si se tiene claro donde se va.

data:

configManagementPlugins: |-

- name: argocd-vault-plugin

generate:

command: ["argocd-vault-plugin"]

args: ["generate", "./"]

**gitops-operator no editable**

Al editar el configMap argocd-cm tendremos de nuevo el agradable aviso de que no es posible al ser un recurso manejado.  
Editaremos el recurso manejante y situaremos la información así:

apiVersion: argoproj.io/v1alpha1

kind: ArgoCD

spec:

configManagementPlugins: |-

- name: argocd-vault-plugin

generate:

command: ["argocd-vault-plugin"]

args: ["generate", "./"]

**Validación de la disponibilidad del plugin**

Para validar la disponibilidad del plugin:

* Rearrancar el reposerver (llevar a cero instancias, y el manejador subirá a 1 automágicamente).
* Entrar a la UI del servidor.
  + Crear una aplicación.
  + Scroll down hasta pasar los grupos general, source, destination
  + El desplegable del último grupo seleccionar Plugins
  + En la sección de Plugins encontrar el ArgoCD Vault Plugin en el desplegable de nombres.

**Deprecación de los argocd-cm CMP**

El proyecto ArgoCD tiene la intención de quitar el soporte de  
los Config Management Plugins (CMP) que usan configmap argocd-cm  
en favor del uso de sidecars (contenedores a los que se les manda un tarball con los manifiestos manejados y que devuelven otro tarball).

* <https://github.com/argoproj/argo-cd/issues/8117>
* <https://argo-cd.readthedocs.io/en/stable/operator-manual/config-management-plugins/>

# Autenticación y autorización por approle en Vault

Esta página propone ejecutar la parte de creación de autentiación y autorización en Vault con approle,  
de forma independiente de otras tareas para centrarse y evitar confusiones.

El role y la policy de Vault creadas se utilizarán en el último paso.

## Documentación de referencia

Referencias sobre authn por approle. El tutorial incluye las policies..

* <https://itnext.io/argocd-secret-management-with-argocd-vault-plugin-539f104aff05>
* <https://developer.hashicorp.com/vault/docs/auth/approle>
* <https://developer.hashicorp.com/vault/tutorials/auth-methods/approle>
* <https://github.com/werne2j/arogcd-vault-plugin-demo>

Similar, pero para authn por kubernetes (service account y RBAC)

* <https://cloud.redhat.com/blog/how-to-use-hashicorp-vault-and-argo-cd-for-gitops-on-openshift>

La última versión de todo es el Vault Agent, con inyección de secretos, mutating webhooks y otras exquisiteces de implementación no trivial.

* <https://cloud.redhat.com/blog/integrating-hashicorp-vault-in-openshift-4>
* <https://developer.hashicorp.com/vault/docs/platform/k8s/injector>
* <https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-sidecar>

## Ejecucion tutorial

Se sigue el tutorial de la lista anterior:

* <https://developer.hashicorp.com/vault/tutorials/auth-methods/approle>

Se trabaja en el terminal del pod vault-0 del statefulset correspondiente.

### Variables de entorno para un vault en local

Skip, ya estamos en el terminal de vault-0. Queda la duda de por qué se anotó este par de valores.

export VAULT\_ADDR=http://127.0.0.1:8200

export VAULT\_TOKEN=root

### Tareas previas como administrador en Vault

Sin estar recogidos en en tutorial, es necesario para evitar un 403/404 en paso siguiente.

Si no se dispone del token, habrá que rearrancar el Vault para que muestre el token y las llaves de desellado (unsealing). Ver el documento de instalación de Vault.

vault login hvs.MbBT9J9ZXyGvlJCjV8CG1IjD

El path enabled -path=secret es solo el directorio raíz del path de los secretos que se quieran crear.

vault secrets enable -path=secret kv-v2

### Crear un secreto para el test

1. Create some test data.

Atención al Secret Path en la respuesta, donde se inserta el conocido /data detrás del path habilitado.

vault kv put secret/mysql/webapp db\_name="users" username="admin" password="passw0rd"

====== Secret Path ======

secret/data/mysql/webapp

======= Metadata =======

Key Value

--- -----

created\_time 2023-08-03T08:05:36.718175618Z

custom\_metadata <nil>

deletion\_time n/a

destroyed false

version 1

### Habilitar approle como método de autenticación en Vault

*Step 1: Enable AppRole auth method (Persona: admin)*

*Enable approle auth method by executing the following command.*

El siguiente comando ya estaba ejecutado de anteriores pruebas.

vault auth enable approle

### Crear un role con una policy de acceso al secreto

*Step 2: Create a role with policy attached  
(Persona: admin)*

#### Crear la policy de acceso al secreto

*When you enabled the AppRole auth method, it gets mounted at the /auth/approle path. In this example, you are going to create a role for the app persona (jenkins in our scenario).  
First, create a policy named jenkins with following definition.*

Ejemplo de definición de policy, no ejecutar:

# Read-only permission on secrets stored at 'secret/data/mysql/webapp'

path "secret/data/mysql/webapp" {

capabilities = [ "read" ]

}

Ahora si, aplicar la definición anterior de policy que permite leer el secreto en el path de sus datos (incluye /data/).

*Before creating a role, create a jenkins policy.*

vault policy write jenkins -<<EOF

# Read-only permission on secrets stored at 'secret/data/mysql/webapp'

path "secret/data/mysql/webapp" {

capabilities = [ "read" ]

}

EOF

Success! Uploaded policy: jenkins

#### Creación del role con la policy anterior

*Creates a role named jenkins with jenkins policy attached. The generated token's time-to-live (TTL) is set to 1 hour and can be renewed for up to 4 hours of its first creation. (NOTE: This example creates a role which operates in pull mode.)*

Interesa el token\_policies que relaciona con la política de lectura anterior.  
Somos generosos con el TTL, en lugar de horas.

vault write auth/approle/role/jenkins \

token\_policies="jenkins" \

token\_ttl=5d token\_max\_ttl=30d

Success! Data written to: auth/approle/role/jenkins

*Read the jenkins role you created to verify.*

vault read auth/approle/role/jenkins

Key Value

--- -----

...

token\_max\_ttl 720h

token\_policies [jenkins]

token\_ttl 120h

### Obtención de RoleID y SecretID

*Step 3: Get RoleID and SecretID*

*The RoleID and SecretID are like a username and password that a machine or app uses to authenticate.  
Since the example created a jenkins role which operates in pull mode, Vault will generate the SecretID. You can set properties such as usage-limit, TTLs, and expirations on the SecretIDs to control its lifecycle.  
To retrieve the RoleID, invoke the auth/approle/role/<ROLE\_NAME>/role-id endpoint. To generate a new SecretID, invoke the auth/approle/role/<ROLE\_NAME>/secret-id endpoint.*

#### Obtención de RoleID

Primero obtener el role\_id, el equivalente al "nombre de usuario" del role para las operaciones de autenticación. El nombre que le hemos dado sería el nombre descriptivo.

*Now, you need to fetch the RoleID and SecretID of a role.  
Execute the following command to retrieve the RoleID for the jenkins role.*

vault read auth/approle/role/jenkins/role-id

Key Value

--- -----

role\_id 9fa0a37a-d130-0b6c-8b30-a18fb203dc10

#### Obtención de SecretID

Luego de tener el role\_id se puede obtener el secret\_id.  
En las operaciones de autenticación este sería la password.

Importante notar que tiene una validez máxima de 30 días y que se debe hacer una renovación periódica por política de seguridad.

*Execute the following command to generate a SecretID for the jenkins role.*

vault write -force auth/approle/role/jenkins/secret-id

Key Value

--- -----

secret\_id 8f87d184-23e3-2cc0-b6bc-ca63478f83b7

secret\_id\_accessor e0010d48-ce7f-e717-d47d-912d0150eb7a

secret\_id\_num\_uses 0

secret\_id\_ttl 0s

*The -force (or -f) flag forces the write operation to continue without any data values specified. Or you can set parameters such as cidr\_list.  
If you specified secret\_id\_ttl, secret\_id\_num\_uses, or bound\_cidr\_list on the role in Step 2, the generated SecretID carries out the conditions.*

*Tip: The RoleID is similar to a username; therefore, you will get the same value for a given role. In this case, the jenkins role has a fixed RoleID. While SecretID is similar to a password that Vault will generate a new value every time you request it.*

### Verificación anteriores pasos

#### Login con RoleID y SecretID para obtener token de acceso

*Step 4: Login with RoleID & SecretID  
(Persona: app)*

*To login, use the auth/approle/login endpoint by passing the RoleID and SecretID.  
Example:*

vault write auth/approle/login \

role\_id="9fa0a37a-d130-0b6c-8b30-a18fb203dc10" \

secret\_id="8f87d184-23e3-2cc0-b6bc-ca63478f83b7"

Key Value

--- -----

token hvs.CAESIIulXqsaZSsGj1DwRPeEp-g2Ssq1qa3OA04wPQp9LVLIGh4KHGh2cy50QXVTSUJIMzcxRDVEbHl1MEpGSElDdVM

token\_accessor LnCeSXHQux3PmtMZ0MrqIIDD

token\_duration 120h

token\_renewable true

token\_policies ["default" "jenkins"]

identity\_policies []

policies ["default" "jenkins"]

token\_meta\_role\_name jenkins

*Vault returns a client token with default and jenkins policies attached.  
Store the generated token value in an environment variable named, APP\_TOKEN.  
Example:*

export APP\_TOKEN="hvs.CAESIIulXqsaZSsGj1DwRPeEp-g2Ssq1qa3OA04wPQp9LVLIGh4KHGh2cy50QXVTSUJIMzcxRDVEbHl1MEpGSElDdVM"

#### Leer secreto con el token devuelto por el login

*Step 5: Read secrets using the AppRole token  
(Persona: app)*

Ejecutar en una sola línea:

VAULT\_TOKEN=$APP\_TOKEN vault kv get secret/mysql/webapp

====== Secret Path ======

secret/data/mysql/webapp

======= Metadata =======

Key Value

--- -----

created\_time 2023-08-03T08:05:36.718175618Z

custom\_metadata <nil>

deletion\_time n/a

destroyed false

version 1

====== Data ======

Key Value

--- -----

db\_name users

password passw0rd

username admin

Si se llama directamente a vault kv get secret/mysql/webapp se usa el login general en vault de la sesión bash.  
Al poner delante VAULT\_TOKEN=$APP\_TOKEN si se tiene en cuenta ese valor específico.

VAULT\_TOKEN=$APP\_TOKEN vault kv delete secret/mysql/webapp

Error deleting secret/data/mysql/webapp: Error making API request.

URL: DELETE http://127.0.0.1:8200/v1/secret/data/mysql/webapp

Code: 403. Errors:

\* 1 error occurred:

\* permission denied

## Documentación adicional sobre role\_id y secret\_id

Del recurso <https://developer.hashicorp.com/vault/docs/auth/approle>

**Credentials/Constraints**

*RoleID is an identifier that selects the AppRole against which the other credentials are evaluated. When authenticating against this auth method's login endpoint, the RoleID is a required argument (via role\_id) at all times. By default, RoleIDs are unique UUIDs, which allow them to serve as secondary secrets to the other credential information. However, they can be set to particular values to match introspected information by the client (for instance, the client's domain name).*

*SecretID is a credential that is required by default for any login (via secret\_id) and is intended to always be secret. (For advanced usage, requiring a SecretID can be disabled via an AppRole's bind\_secret\_id parameter, allowing machines with only knowledge of the RoleID, or matching other set constraints, to fetch a token). SecretIDs can be created against an AppRole either via generation of a 128-bit purely random UUID by the role itself (Pull mode) or via specific, custom values (Push mode). Similarly to tokens, SecretIDs have properties like usage-limit, TTLs and expirations.*

**Pull and push SecretID modes**

*If the SecretID used for login is fetched from an AppRole, this is operating in Pull mode. If a "custom" SecretID is set against an AppRole by the client, it is referred to as a Push mode. Push mode mimics the behavior of the deprecated App-ID auth method; however, in most cases Pull mode is the better approach. The reason is that Push mode requires some other system to have knowledge of the full set of client credentials (RoleID and SecretID) in order to create the entry, even if these are then distributed via different paths. However, in Pull mode, even though the RoleID must be known in order to distribute it to the client, the SecretID can be kept confidential from all parties except for the final authenticating client by using Response Wrapping.*

*Push mode is available for App-ID workflow compatibility, which in some specific cases is preferable, but in most cases Pull mode is more secure and should be preferred.*

**Further constraints**

*role\_id is a required credential at the login endpoint. AppRole pointed to by the role\_id will have constraints set on it. This dictates other required credentials for login. The bind\_secret\_id constraint requires secret\_id to be presented at the login endpoint. Going forward, this auth method can support more constraint parameters to support varied set of Apps. Some constraints will not require a credential, but still enforce constraints for login. For example, secret\_id\_bound\_cidrs will only allow logins coming from IP addresses belonging to configured CIDR blocks on the AppRole.*

# Despliegue de aplicaciones en ArgoCD con Vault approle

En este paso final se conecta la autenticación por approle de Vault con las credenciales de configuración del ArgoCD Vault Plugin (AVP).

En resumen:

* Las credenciales necesarias se ponen en un secreto de OpenShift.
* Se modifica el repo server para que tome variables de entorno a partir del secreto anterior.  
  En estas variables va la configuración de AVP.

## Rotación de secret\_id

De cara a la rotación de secret\_id, de los approle de Vault, será necesario poder periódicamente:

* Acceder a Vault para hacer un write -f que genere un nuevo secret\_id (por API de Vault parece el mejor método).
* Actualizar el secreto (API de OpenShift).
* Rearrancar el repo server de ArgoCD para aplicar los cambios, como rollout restarts.

Muy por encima, secretos y configmaps no son versionables en K8s, lo que hace difícil forzar rearranques. Pero esto también evita que configuraciones no validadas rearranquen un proyecto.

A futuro, este artículo propone realizar los rollout restarts  
uniendo configMap o Secret con el deployment o statefulset  
anotando en el YAML de los segundos el hash de los primeros.  
Esto se podrá hacer de forma periódica y OpenShift/K8s sólo hará rearranques cuando haya un cambio verdadero.

* <https://blog.questionable.services/article/kubernetes-deployments-configmap-change/>

## Antes de empezar

Tomamos como fuente

* <https://itnext.io/argocd-secret-management-with-argocd-vault-plugin-539f104aff05>

Y suponemos al lector habituado con la guía anterior sobre approle, role\_id y secret\_id.  
De esta aprovecharemos el approle jenkins y su política de acceso a un secreto en un path.  
Pero usaremos otro secreto en otro path y añadiremos ese path a la política.

## Variables de entorno para configurar el AVP

### Conseguir role\_id y secret\_id

Ir al namespace myargocd  
y abrir un terminal del repo server  
donde podemos ejecutar el argocd-vault-plugin.  
De una instalación anterior a esta prueba de concepto, tenemos en namespace myargocd en la instancia de myargocd-repo-server:  
Los valores de VAULT\_ADDR, AVP\_TYPE y AVP\_AUTH\_TYPE son correctos.

echo $VAULT\_ADDR && echo $AVP\_TYPE && echo $AVP\_AUTH\_TYPE && echo $AVP\_ROLE\_ID && echo $AVP\_SECRET\_ID

http://vault.vault-infra.svc.cluster.local:8200

vault

approle

{role\_id inadecuado}

{secret-id desfasado}

Vamos al namespace y pod de vault y  
usamos lo descrito para obtener un secret-id para el role\_id definido.

vault login hvs.MbBT9J9ZXyGvlJCjV8CG1IjD

Success! ---

vault read auth/approle/role/jenkins/role-id

role\_id 9fa0a37a-d130-0b6c-8b30-a18fb203dc10

vault write -force auth/approle/role/jenkins/secret-id

secret\_id 25cf360f-3171-74c6-5cf0-f71cbc28d729

secret\_id\_accessor 4bd420b4-2f19-264a-7575-2f0931f3b0fa

secret\_id\_num\_uses 0

secret\_id\_ttl 0s

### Paso de variables de entorno por Secret

El repo-server carga los valores como variables de entorno  
desde un secret argocd-vault-plugin-credentials que vamos a crear.

Aplicar este fichero con los valores oportunos pasados por el codificador base64.  
Para ofuscar un valor en Base64 **NO USAR BAJO NINGÚN CONCEPTO**  
este comando, que añade un \n final  
echo "Valor a ofuscar" | base64.  
Esto causó horrores intentando hacer que funcionase la conexión de AVP a Vault.

Fichero a aplicar en el namespace myargocd con todos los valores en base64.

kind: Secret

apiVersion: v1

metadata:

name: argocd-vault-plugin-credentials

namespace: argocd

type: Opaque

data:

AVP\_AUTH\_TYPE: approle

AVP\_TYPE: vault

AVP\_ROLE\_ID: your\_role\_id

AVP\_SECRET\_ID: your\_secret\_id

VAULT\_ADDR: your\_vault\_addr

Una vez aplicados los valores ofuscados se pueden ver los valores cargados revelándolos  
en el apartado de Secrets del namespace myargocd.

Para que el repo-server cargue los valores en este Secret como variables de entorno,  
ir a Deployments -> Repo Server y en la pestaña de Environment  
usar All values from existing ConfigMaps or Secrets (envFrom) ConfigMap/Secret.  
Esto escribe el envFrom dentro de una de las instancias de containers (a la altura de las sondas y volume mounts) y allí se respeta.  
No se puede editar manualmente en el YAML.

El despliegue de repo-server necesita un reinicio para adquirir las variables de entorno en un nuevo pod.

#### Otras formas de definir las variables de entorno

La forma de definir las variables de entorno en el repo server es la más funcional encontrada.

Anteriormente, en la fuente del tutorial se observaba:

*The only thing left to do is to make that secret available as environment variables in the argocd-repo-server pods. Go back to the argocd-repo-server deployment where you initially added the initContainer and add an envFrom pointing to the secret you just created.*

containers:

- name: argocd-repo-server

volumeMounts:

- name: custom-tools

mountPath: /usr/local/bin/argocd-vault-plugin

subPath: argocd-vault-plugin

envFrom:

- secretRef:

name: argocd-vault-plugin-credentials

Cuidado que si los despliegues de ArgoCD están gestionados por \_openshif\_gitops\_ habrá que definirlo en el YAML de su manejador, el ACD myargocd:

spec:

repo:

envFrom:

- secretRef:

name: argocd-vault-plugin-credentials

*That is it! You should now have a configured plugin pointing to your Vault. We are now ready to test out the plugin!*

Pues no. La entrada envFrom se borra y desaparece del YAML si se pretende meter en el YAML. Curiosamente, si que se conserva el envFrom si se pone en el initContainer, pero pasar variables de entorno desde un initContainer requiere soluciones demasiado creativas.

Repasando el yaml que define el yaml de este CRD caemos en cuenta de que se puede definir como variables individuales (!)  
<https://github.com/redhat-developer/gitops-operator/blob/master/config/crd/bases/argoproj.io_argocds.yaml>

repo:

env:

- name: VAULT\_ADDR

valueFrom:

secretKeyRef:

name: argocd-vault-plugin-credentials

key: VAULT\_ADDR

Este método funciona pero no es tan compacto como el propuesto en primer lugar.

## Creación del secreto

Para desplegar una aplicación de ejemplo es necesario crear  
otro secreto en el path avp/test, lo que requiere los ya conocidos:

vault secrets enable -path=avp kv-v2

vault kv put avp/test sample=secret

== Secret Path ==

avp/data/test

======= Metadata =======

Key Value

--- -----

created\_time 2023-08-07T05:57:40.32059702Z

custom\_metadata <nil>

deletion\_time n/a

destroyed false

version 1

vault kv get avp/test

== Secret Path ==

avp/data/test

===== Data =====

Key Value

--- -----

sample secret

Es necesaria una policy para leer avp/test.  
Reciclamos la policy jenkins ya asociada al role homónimo

<https://vaultinfra.apps.k8spro.nextret.net/ui/vault/policy/acl/jenkins>

# Read-only permission on secrets stored at 'secret/data/mysql/webapp'

path "secret/data/mysql/webapp" {

capabilities = [ "read" ]

}

# Más acceso para esta política.

path "avp/data/test" {

capabilities = [ "read" ]

}

No se ha encontrado la forma de modificar la política desde CLI.

## Configuración aplicación ArgoCD

Usaremos la aplicación ejemplo definida en <https://github.com/werne2j/arogcd-vault-plugin-demo>  
donde example-secret.yaml contiene:

* Una marca <sample> como el sample definido para el secreto en Vault.
* La anotación avp.kubernetes.io/path: "avp/data/test" que coincide con el secret path devuelto para el secreto en Vault.

kind: Secret

apiVersion: v1

metadata:

name: example-secret

annotations:

avp.kubernetes.io/path: "avp/data/test"

type: Opaque

stringData:

sample-secret: <sample>

A través de la ruta myargocd-server llegamos a <https://myargocd-server-myargocd.apps.k8spro.nextret.net>

Para obtener la password del usuario

oc extract secret/myargocd-cluster -n myargocd --to=-

# admin.password

5JmxFBg23eAcizLHvO0fNYEbdRPjXTq8

Y a partir de aquí crear una aplicación tal como indica el tutorial de referencia.

*Open ArgoCD and create a new application*

*We are going to name it sample-secret and put it in the default project*

*I have a sample repo that we will use to pull a example secret file from at* [*https://github.com/werne2j/arogcd-vault-plugin-demo*](https://github.com/werne2j/arogcd-vault-plugin-demo)

*We will put the secret in-cluster (Within the cluster ArgoCD is installed) and in the default namespace*

*The last piece needed is to specify the argocd-vault-plugin plugin to be used*

*Now we can click the create button and see if it worked!*

*You should see an application created in the ArgoCD UI*

*And if you click the application, you will hopefully see this:*

*If so, you have successfully used the argocd-vault-plugin! We can confirm this by looking for the secret in Kubernetes and checking its value:*

El tutorial sugiere aquí refrescar el valor en Vault y volver a sincronizarlo.  
Para el refresco con el hard refresh ver la siguiente sección **Sincronización**

*However we are not done yet! One of the great things about the plugin is that if the value changes in Vault we can update the value in the cluster with little effort. So update the value in vault:*

vault kv put avp/test sample=new\_secret

*Now in ArgoCD you can do a hard refresh, this will perform a dry-run of the plugin*

*Now you should notice that is application is out of sync:*

*This means that the plugin performed the dry run and determined that the output was different than what was currently in the cluster. Now all we have to do is sync the application and we should see the application back green!*

## Despliegue en otros namespaces

A efectos de un despliegue en ArgoCD hay que contar con un conjunto de ámbitos:

* El **clúster**, que en esta prueba de concepto se restringe al clúster local.
* El **proyecto ArgoCD**, que es una entidad que tiene sentido en el ámito de ArgoCD.
* El **namespace** o **project** de **OpenShift**.  
  Es la entidad que agrupa recursos de forma lógica en OpenShift.  
  Hay que tener cuidado de no confundirlo con el proyecto de ArgoCD.

Con una instancia de ArgoCD fresca y sin más configuración  
hay una limitación importante: Sólo se puede desplegar al propio namespace de ArgoCD. En esta prueba de concepto, myargocd.  
Si al crear una app de ArgoCD se le da como destino un namespace sin preparar nada, tendremos el error:

ComparisonError

Namespace "mynamespace"

for Secret "example-secret"

is not managed

En <https://access.redhat.com/solutions/6158462>  
se proponen dos soluciones en función de la granularidad que se quiera dar del control sobre el namespace destino.

### Control total del namespace destino

Esta solución consiste en crear (borrar si ya existe)  
el namespace de forma que sea gestionado por ArgoCD.  
Esto implica que ArgoCD puede hacer cualquier cosa en el namespace.

cat << EOF >> nstest.yaml

apiVersion: v1

kind: Namespace

metadata:

name: mynamespace

labels:

argocd.argoproj.io/managed-by: myargocd

EOF

oc apply -f nstest.yaml

namespace/mynamespace created

Elementos a sustituir en el manifiesto:

* name: mynamespace: new namespace to be managed by an existing Argo CD instance.
* argocd.argoproj.io/managed-by: myargocd: namespace where Argo CD is deployed.

Con esto ha sido posible deplegar la aplicación al nuevo namespace manejado por ArgoCD.

### Control parcial del namespace destino

Se deja aquí anotada la opción que da control a ArgoCD con una granularidad fina de permisos, ya que el artículo requiere una cuenta de RedHat para acceder a leerlo.

*There are two options to grant permissions to ArgoCD  
to manage different namespaces. First option applyes roles and rolebindings  
and it is referenced in another solution (*[*https://access.redhat.com/solutions/5875661)*](https://access.redhat.com/solutions/5875661))*.  
This solutions adds more flexibility as the resources and verbs  
that are granted can be defined.*

*It is needed to create a Role and a RoleBinding to give privileges to the new ArgoCD instance. This way, ArgoCD will be able to create resources in a different namespace.*

*The role.yaml should be similar to the one below, but it can be configured with different verbs depending on the user case:*

$ cat << EOF >> role.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: <role-name>

namespace: <target-ns>

rules:

- apiGroups:

- "\*"

resources:

- "\*"

verbs:

- Get

- List

- Watch

- Patch

EOF

$ oc create -f role.yaml

The RoleBinding should be similar to the following:

$ cat << EOF >> rolebinding.yaml

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: RoleBinding

metadata:

name: <role-binding-name>

namespace: <target-ns>

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: <role-name>

subjects:

- kind: ServiceAccount

name: <application-controller-name>

namespace: <application-controller-namespace>

EOF

oc create -f rolebinding.yaml

Valores a sustituir:

* role-binding-name: Nombre a voluntad para el role binding.
* role-name: Nombre a voluntad para el role.
* target-ns: El namespace a controlar ya creado.
* application-controller-name: Suponemos para la prueba de concepto el myargocd-application-controller.
* application-controller-namespace: Suponemos para la prueba de concepto el myargocd.

*After creating the Role and RoleBindig, the sync process should finish successfully.*

No lo hace pero por la necesidad ya indicada del tutorial de hacer un hard refresh para el caso de los secretos.

## Creación de objetos para desplegar una app en ArgoCD

Se ha visto como es necesario crear el namespace destino de forma que sea controlado por ArgoCD para poder desplegar.

Del mismo modo, el proyecto de ArgoCD también tiene algunas sutilidades. Veamos un modo conveniente de crearlos.  
Los objetos creados de forma conveniente tienen una ventaja: Aparecen en los desplegables de la página de creación de una app en la UI de ArgoCD.

### Creación del proyecto

El proyecto se puede crear sin más en la UI de ArgoCD.

### Registrar el repositorio git

Para repositorios públicos esto no es necesario, pero para repos privados protegidos con una clave privada lo siguiente es necesario. Y de paso y como se ha dicho, aparece en el desplegable de repositorios.

Se recomienda tener un mínimo de repositorios de configuración de ArgoCD y luego usar el path relativo a la raíz. En la prueba de concepto se han usado repos específicos con . como path para usarlos directamente en su raíz.

De otro proyecto en otra entidad tomamos este ejemplo.

La autenticación se hace por una clave privada que tendremos en un fichero de texto llamado en este ejemplo id\_rsa

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQE(…)

(…)OZmfh6DQ3D6pTGDPwxFFhX5sLrwshSgu11bW9

-----END RSA PRIVATE KEY-----

Se añade el URL de git repo el proyecto.

argocd repo add \

git@gitlab.irtve.rtve.int:kubernetes/rtve-app-pf-play.git

Repository 'git@gitlab.irtve.rtve.int:kubernetes/rtve-app-pf-play.git' added

Si el repositorio es privado y no se registra debidamente en ArgoCD, cuando se cree la aplicación se obtendrá un error como el siguiente:

Unable to create application: application spec for pf-news is invalid:

InvalidSpecError: application repo git@gitlab.irtve.rtve.int:kubernetes/rtve-app-pf-news.git is not permitted in project 'rtve-pro'

### Permisos en el namespace destino

Esta es una solución alternativa a las vistas anteriormente, que funcionó con el gitops-operator, no en una instancia personalizada.

Para que un proyecto de ArgoCD pueda desplegar en un namespace de Openshift es necesario ejecutar el comando argocd proj add-destination <PROJECT> <CLUSTER>,<NAMESPACE>, que permite que el namespace de Openshift sea destino de las aplicaciones en el proyecto ArgoCD. Esta operación es necesaria una sola vez para todas las aplicaciones en ese proyecto.

argocd proj add-destination \

rtve-pro \

https://kubernetes.default.svc \

rtve-pf

Si no se ha permitido que el proyecto ArgoCD despliegue en el namespace de OpenShift, cuando se cree la aplicación se obtendrá un error como el siguiente:

Unable to create application: application spec for pf-news is invalid:

InvalidSpecError: application destination {https://kubernetes.default.svc rtve-pf} is not permitted in project 'rtve-pro'

Para disponer de los permisos de creación de aplicaciones en un namespace de OpenShift se eleva (previamente) el serviceaccount de ArgoCD a admin en Openshift.

>oc adm policy add-role-to-user admin system:serviceaccount:openshift-gitops:openshift-gitops-argocd-application-controller -n rtve-pf

La falta de esta elevación podría haber dado lugar a un mensaje como este

Unable to create application: permission denied: applications, create, rtve-pro/pf-news, sub: CiRhMzk4NDliZC05ZWQxLTRlMWUtYTdiMi1kOTcyOWRlMDEwNmYSCW9wZW5zaGlmdA, iat: 2022-12-14T08:50:39Z

## Refresco y deprecación del AVP

El tutorial seguido habla de la necesidad de hacer un hard refresh y luego sincronizar.  
El hard refresh es una subopción disponible en el botón del refresh de la UI,  
y también se puede hacer desde línea de comandos.  
Después del hard refresh, por el método que sea, el sync de la aplicación ArgoCD trae los nuevos valores del Vault.

La explicación plausible es que el secreto fuente en ArgoCD no cambia, tiene la misma marca de sustitución.  
Al hacer el hard refresh se hace un dry run en el que ArgoCD ve que la salida del plugin es distinta de lo que tiene,  
y es en ese momento cuando ArgoCD considera que está OutOfSync y ejecutará el siguiente sync realmente.

argocd app get sample-secrets-mynamespace --hard-refresh

time="2023-08-09T11:03:35+02:00" level=warning msg="spec.plugin.name is set, which means this Application uses a plugin installed in the argocd-cm ConfigMap. Installing plugins via that ConfigMap is deprecated in Argo CD v2.5. Starting in Argo CD v2.6, this Application will fail to sync. Contact your Argo CD admin to make sure an upgrade plan is in place. More info: https://argo-cd.readthedocs.io/en/latest/operator-manual/upgrading/2.4-2.5/"

Name: myargocd/sample-secrets-mynamespace

Project: myargocdproject

Server: https://kubernetes.default.svc

Namespace: mynamespace

URL: https://myargocd-server-myargocd.apps.k8spro.nextret.net/applications/sample-secrets-mynamespace

Repo: https://github.com/werne2j/arogcd-vault-plugin-demo.git

Target: HEAD

Path: .

SyncWindow: Sync Allowed

Sync Policy: <none>

Sync Status: OutOfSync from HEAD (b7cec5a)

Health Status: Healthy

GROUP KIND NAMESPACE NAME STATUS HEALTH HOOK MESSAGE

Secret mynamespace example-secret OutOfSync secret/example-secret configured

oc extract secret/example-secret -n mynamespace --to=-

# sample-secret

valor nuevo

Recordatorio:

oc extract secret/myargocd-cluster -n myargocd --to=-

argocd login myargocd-server-myargocd.apps.k8spro.nextret.net

Respecto al mensaje que aparece en el hard refresh:

level=warning

msg="spec.plugin.name is set, which means this Application

uses a plugin installed in the argocd-cm ConfigMap.

Installing plugins via that ConfigMap is deprecated in Argo CD v2.5.

Starting in Argo CD v2.6, this Application will fail to sync.

Contact your Argo CD admin to make sure an upgrade plan is in place. More info: https://argo-cd.readthedocs.io/en/latest/operator-manual/upgrading/2.4-2.5/"

De hecho estar, ya estamos en ArgoCD 2.6 sever y no ha fallado...

argocd version

argocd: v2.5.3+0c7de21

BuildDate: 2022-11-28T17:22:28Z

GitCommit: 0c7de210ae66bf631cc4f27ee1b5cdc0d04c1c96

GitTreeState: clean

GoVersion: go1.18.8

Compiler: gc

Platform: windows/amd64

>> argocd-server: v2.6.7+unknown

En cualquier caso, no es nuevo el tema de deprecar la configuración  
del plugin AVP con el argocd-cm configMap y sería bueno  
plantear el uso de contenedores sidecar para este cometido.

* <https://github.com/argoproj/argo-cd/issues/11689>
  + <https://argo-cd.readthedocs.io/en/stable/user-guide/config-management-plugins/>
    - <https://argo-cd.readthedocs.io/en/stable/operator-manual/config-management-plugins/>