Resumen de la instalación de ArgoCD Vault Plugin (AVP) en clúster piloto Openshift

- Install AVP.

Install AVP

Vault

Install Vault confunde que en la doc está la fallida de segun Openshift

```
oc create namespace vault-infra
git clone https://github.com/hashicorp/vault-helm.git
```

Configurar la instalación de Vault con Helm

```
cd vault-helm
vi values.yaml
    global:
        # If deploying to OpenShift
        openshift: true
    server:
        route:
            enabled: true
    ui:
        # True if you want to create a Service entry for the Vault
UI.
        #
        # serviceType can be used to control the type of service
created. For
        # example, setting this to "LoadBalancer" will create an
external load
        # balancer (for supported K8S installations) to access the
UI.
        enabled: true
```

Cambiar los `chart-example.local` por `vaultinfra.apps.ocp-pro.infra.msc.es`.

```
  ingress:
    enabled: false
    hosts:
      - host: vaultinfra.apps.ocp-pro.infra.msc.es
  # 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.ocp-pro.infra.msc.es
```

Instalar Vault a partir del Helm Chart moldificado

```
helm install vault . -n vault-infra
```

Está pendiente de inicializar

```
oc get pods -n vault-infra
```

Desde el terminal del pod vault-0 (ir al statefulset y buscar sus pods).

```
vault version
vault operator init
    Unseal Key 1: j/a77Mqs6O1r7YAAf8yGjaGLN+9I8n44caqiOaTS01CH
    Unseal Key 2: A2fCUz5mRPSwixzr2CzG9zyQ2ZhxxspnkgMHv4RIqU/U
    Unseal Key 3: CL9P2pOzK8S5VazgaAEAYYkyUHTDubjxGyfG4oayAAbp
    Unseal Key 4: J2O9NR+IBtvAt8esBGEBLMPBJYFdHQgEkJzy38gp1s98
    Unseal Key 5: zY9yFxZzsO6/SwQwA+JE0xjdI0150KNwAD6XndArAxm7

    Initial Root Token: hvs.CXc358eUqb1bhvMOz94hRMUA


vault operator unseal j/a77Mqs6O1r7YAAf8yGjaGLN+9I8n44caqiOaTS01CH
vault operator unseal A2fCUz5mRPSwixzr2CzG9zyQ2ZhxxspnkgMHv4RIqU/U
vault operator unseal CL9P2pOzK8S5VazgaAEAYYkyUHTDubjxGyfG4oayAAbp

vault login hvs.CXc358eUqb1bhvMOz94hRMUA
    Key                  Value
    ---                  -----
    token                hvs.CXc358eUqb1bhvMOz94hRMUA
    token_accessor       yyKsNtErNIs7f3CXSz5VzP1Y
    token_duration       ∞
    token_renewable      false
    token_policies       ["root"]
    identity_policies    []
    policies             ["root"]
vault secrets enable -path=ceh kv-v2
vault kv put ceh/database/credentials \
  username="db-username" \
  password="db-password"
 vault kv get ceh/database/credentials
```

Cambiar en la ruta del namespace de
`spec.tls.termination: passthrough`
a
`spec.tls.termination: edge`

Acceder a https://vaultinfra.apps.ocp-pro.infra.msc.es/ y autenticar con el root token. Buscar el secreto creado.

Instancia personalizada de ArgoCD

Crear namespace

```
oc new-project myargocd
```

Seleccionar namespace y con el icono (+) de consola desplegar YAML:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: myargocd
spec:
  server:
    route:
      enabled: true
```

Extracción de password de admin

```
oc get secret myargocd-cluster -n myargocd -
ojsonpath='{.data.adminyargocd -ojsonpath='{.data.admin\.password}'
| base64 -d
p9KEFGmO0Uo7nJ2NwPxqHgd3bYSTfZDh
(tambien, y funciona en Windows)
oc extract secret/myargocd-cluster -n myargocd --to=-
```

https://myargocd-server-myargocd.apps.ocp-pro.infra.msc.es/

Instalación de ArgoCD Vault plugin

El plugin se descarga de momento de github. Más adelante se planteará un repositorio HTTP propio para artefactos descargables.
Para salir al exterior es necesario pasar por el proxy `proxy-sanidad.msc.es:8080` (autenticado).
Para el `initContainer` que descarga el plugin se crea una imagen base `harbor.apps.ocp-pro.infra.msc.es/base_images/alpine-curl` con este Dockerfile.

```
FROM alpine:3.8

RUN apk update
RUN apk upgrade
RUN apk add curl
```

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 spec.repo: {} por esta estructura con
volumes, volumeMounts e initContainers:

```
  repo:
    initContainers:
      - args:
          - >-
            cd /tmp

            curl --proxy http://APP.GUNIX-
ADM:n8a2VK98PKs8AGUOTxRzS8qN@proxy-sanidad.msc.es:8080 --noproxy
.msc.es,.sanidad.msc -sSfL -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/
        command:
          - sh
          - '-c'
        image: 'harbor.apps.ocp-pro.infra.msc.es/base_images/alpine-
curl:latest'
        name: download-tools
        volumeMounts:
          - mountPath: /custom-tools
            name: custom-tools
    volumeMounts:
      - mountPath: /usr/local/bin/argocd-vault-plugin
        name: custom-tools
        subPath: argocd-vault-plugin
    volumes:
      - emptyDir: {}
        name: custom-tools
```

Para controlar el arranque del repo server y verificar que no existe ningún mensaje de error en el initContainer `download-tools` (sustituir `74cbcc69b7-wzfkw` por el sufijo del pod real y `myargocd` por el namespace):

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

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:*

Editaremos el recurso que maneja el repo server (el mismo donde se ha definido el initContainer anterior) 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.

```
https://console-openshift-console.apps.ocp-
pro.infra.msc.es/k8s/ns/myargocd/secrets/myargocd-cluster
https://myargocd-server-myargocd.apps.ocp-pro.infra.msc.es/
```

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

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

Referencia

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

Respecto a este recurso, ya ejecutado en una primera prueba, se cambian los siguientes elementos

- La policy `jenkins` se renombra `avptestplcy`.
- El role `jenkins` se renombra `avptestrole`.
- El secreto `secret/mysql/webapp` se creará ya directamente como `avp/test`.

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. El primero es perogrullo y no hará daño, el segundo se sustituirá por otros 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.CXc358eUqb1bhvMOz94hRMUA
```

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

```
vault secrets enable -path=avp kv-v2
```

Crear un secreto para el test

4. Create some test data.

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

```
vault kv put avp/test sample="valor-secreto"
    == Secret Path ==
    avp/data/test

    ======= Metadata =======
    Key                Value
    ---                -----
    created_time       2023-09-15T06:27:44.341058654Z
    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.*

```
vault auth enable approle
    Success! Enabled approle auth method at: 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 (avptestrole in our scenario).*
> *First, create a policy named avptestplcy with following definition.*
> *Before creating a role, create a avptestplcy policy.*

Con el nombre antes elegido `avptestplcy` y el secret path `avp/data/test`:

```
vault policy write avptestplcy -<<EOF
# Read-only permission on secrets stored at 'avp/data/test'
path "avp/data/test" {
  capabilities = [ "read" ]
}
EOF

    Success! Uploaded policy: avptestplcy
```

Creación del role con la policy anterior

> *Creates a role named avptestrole with avptestplcy 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/avptestrole \
    token_policies="avptestplcy" \
    token_ttl=5d token_max_ttl=30d

    Success! Data written to: auth/approle/role/avptestrole
```

> *Read the avptestrole role you created to verify.*

```
vault read auth/approle/role/avptestrole
    Key                     Value
    ---                     -----
    token_max_ttl           720h
    token_ttl               120h
    token_policies          [avptestplcy]
```

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 avptestrole 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 avptestrole role.*

```
vault read auth/approle/role/avptestrole/role-id

    Key        Value
    ---        -----
    role_id    087481cd-188c-35cd-9a78-0743e5cae6e2
```

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 avptestrole role.*

```
vault write -force auth/approle/role/avptestrole/secret-id
    Key                   Value
    ---                   -----
    secret_id             bbe28015-0e62-5445-39fb-97cedc835b95
    secret_id_accessor    e79bb1e4-bb59-8fef-803e-f5c73bbb4a41
    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="087481cd-188c-35cd-9a78-0743e5cae6e2" \
    secret_id="bbe28015-0e62-5445-39fb-97cedc835b95"

        Key                  Value
        ---                  -----
        token
hvs.CAESICcWYezsUud84vP8dhcEFdzJLCM3AyrOzuhF_-
XXeKXfGh4KHGh2cy5LVEZGQlVXT3pNbVpCOXlvYkVSWko1bmU
        token_accessor       VzzSkbv1GctilJKjnGlMiU6z
        token_duration       120h
```

```
token_renewable         true
token_policies          ["avptestplcy" "default"]
identity_policies       []
policies                ["avptestplcy" "default"]
token_meta_role_name    avptestrole
```

> *Vault returns a client token with default and avptestplcy policies attached.*
> *Store the generated token value in an environment variable named, APP_TOKEN.*
> *Example:*

```
export APP_TOKEN="hvs.CAESICcWYezsUud84vP8dhcEFdzJLCM3AyrOzuhF_-
XXeKXfGh4KHGh2cy5LVEZGQlVXT3pNbVpCOXlvYkVSWko1bmU"
```

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 avp/test

    == Secret Path ==
    avp/data/test

    ======= Metadata =======
    Key                Value
    ---                -----
    created_time       2023-09-15T06:27:44.341058654Z
    custom_metadata    <nil>
    deletion_time      n/a
    destroyed          false
    version            1

    ===== Data =====
    Key       Value
    ---       -----
    sample    valor-secreto
```

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 avp/test
    Error deleting avp/data/test: Error making API request.
    URL: DELETE http://127.0.0.1:8200/v1/avp/data/test
    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 `avptestrole` y su política de acceso `avptestplcy` a un secreto en un path `avp/test`.

Variables de entorno para configurar el AVP

Conseguir role_id y secret_id

Ir al namespace y pod de vault y
usamos lo descrito para obtener un secret-id para el role_id definido.

```
vault login hvs.CXc358eUqb1bhvMOz94hRMUA
    Success! ---
vault read auth/approle/role/avptestrole/role-id
    role_id    087481cd-188c-35cd-9a78-0743e5cae6e2
vault write -force auth/approle/role/avptestrole/secret-id
    secret_id              5f749f1f-593f-4426-0525-b3898388c032
    secret_id_accessor     3a492f4b-4832-287b-fbaa-9980f55d08d1
    secret_id_num_uses     0
    secret_id_ttl          0s
```

Nos queda

```
role_id    087481cd-188c-35cd-9a78-0743e5cae6e2
secret_id  5f749f1f-593f-4426-0525-b3898388c032
```

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 el namespace `myargocd`.

Aplicar este fichero con los valores oportunos pasados por el codificador base64.

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

```
kind: Secret
apiVersion: v1
metadata:
  name: argocd-vault-plugin-credentials
  namespace: myargocd
type: Opaque
data:
  AVP_AUTH_TYPE: approle
  AVP_TYPE: vault
  AVP_ROLE_ID: 087481cd-188c-35cd-9a78-0743e5cae6e2
  AVP_SECRET_ID: 5f749f1f-593f-4426-0525-b3898388c032
  VAULT_ADDR: http://vault.vault-infra.svc.cluster.local:8200
```

Que ofuscado en base64 queda

```
kind: Secret
apiVersion: v1
metadata:
  name: argocd-vault-plugin-credentials
  namespace: myargocd
type: Opaque
data:
  AVP_AUTH_TYPE: YXBwcm9sZQ==
  AVP_TYPE: dmF1bHQ=
  AVP_ROLE_ID: MDg3NDgxY2QtMTg4Yy0zNWNkLTlhNzgtMDc0M2U1Y2FlNmUy
  AVP_SECRET_ID: NWY3NDlmMWYtNTkzZi00NDI2LTA1MjUtYjM4OTgzODhjMDMy
  VAULT_ADDR:
aHR0cDovL3ZhdWx0LnZhdWx0LWluZnJhLnN2Yy5jbHVzdGVyLmxvY2FsOjgyMDA=
```

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`
y buscar en el deplegable `argocd-vault-plugin-credentials`. No es necesario un prefijo.

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
del YAML del repo server. 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. Verificar que se da automáticamente el reinicio.

En el namespace `myargocd` abrir un terminal del repo server
donde podemos ejecutar el `argocd-vault-plugin`.

Verificar las variables de entorno introducidas por secret y renicio.

```
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
    087481cd-188c-35cd-9a78-0743e5cae6e2
    5f749f1f-593f-4426-0525-b3898388c032
```

Repaso del secreto y policy de acceso

Validamos la existencia del secreto `avp/test` en el pof `vault-0` del namespace `vault-infra`.

```
vault kv get avp/test
    == Secret Path ==
    avp/data/test
    ===== Data =====
    Key       Value
    ---       -----
    sample    valor-secreto
```

Es necesaria una policy para leer `avp/test`.
Revisamos la política `avptestplcy`

https://vaultinfra.apps.ocp-pro.infra.msc.es/ui/vault/policy/acl/avptestplcy

```
# Read-only permission on secrets stored at 'avp/data/test'
path "avp/data/test" {
  capabilities = [ "read" ]
}
```

También:

```
vault policy read avptestplcy
# Read-only permission on secrets stored at 'avp/data/test'
path "avp/data/test" {
  capabilities = [ "read" ]
}
```

Despliegue en otros namespaces

Para que la instancia de ArgoCD pueda desplegar en otros namespaces y no solo en `myargocd`
se creará primero el namespace `mynamespace` con este manifiesto:

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.

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.

https://raw.githubusercontent.com/werne2j/arogcd-vault-plugin-demo/main/example-secret.yaml

```
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.ocp-pro.infra.msc.es/

Para obtener la password del usuario

```
oc extract secret/myargocd-cluster -n myargocd --to=-
    # admin.password
    p9KEFGmO0Uo7nJ2NwPxqHgd3bYSTfZDh
```

También en https://console-openshift-console.apps.ocp-pro.infra.msc.es/k8s/ns/myargocd/secrets/myargocd-cluster

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*
>
> *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!*

Creación de objetos para desplegar una app en ArgoCD

Repasar la documentación extendida de la prueba de concepto de ArgoCD Vault Plugin para más posibles configuracones.