# Vault

## Vault en Docker

<https://hub.docker.com/_/vault?tab=description>

Tag usado para la creación del documento: “vault:1.1.3”

### Volúmenes:

* /vault/logs: Para registro de logs. Requiere activar la auditoría por archivo.
* /vault/file: Para almacenamiento persistente. Requiere activar almacenamiento por archivo.

### Configuración:

* /vault/config: Usado para almacenar archivos .HCL o .JSON. Requiere establecer la variable de entorno VAULT\_LOCAL\_CONFIG

### Nota importante:

Para evitar que los archivos confidenciales se intercambien en disco, el contenedor intentará bloquear la memoria, para lograr esto a la hora de ejecutar Vault en Docker se debe establecer el parámetro --cap-add=IPC\_LOCK

### Ejecutar Vault en ambiente de Desarrollo

docker run --cap-add=IPC\_LOCK -p 8200:8200 -e 'VAULT\_DEV\_ROOT\_TOKEN\_ID=myroot' vault

* --cap-add=IPC\_LOC: Activar bloqueo de memoria.
* -p 8200:8200: Exponer el puerto 8200 del contenedor.
* -e 'VAULT\_DEV\_ROOT\_TOKEN\_ID=myroot': Establecer variable de entorno VAULT\_DEV\_ROOT\_TOKEN\_ID que configura el token del root.

### Ejecutar Vault en ambiente de “Producción”

docker run --cap-add=IPC\_LOCK -p 8200:8200 -e 'VAULT\_ADDR=http://127.0.0.1:8200' -e 'VAULT\_API\_ADDR=http://127.0.0.1:8200' -e 'VAULT\_LOCAL\_CONFIG={ "listener": [{ "tcp": { "address": "0.0.0.0:8200", "tls\_disable" : 1 } }], "backend": { "file": {"path": "/vault/file"} }, "default\_lease\_ttl": "168h", "max\_lease\_ttl": "720h", "ui": "true"}' vault server

* --cap-add=IPC\_LOC: Activar bloqueo de memoria.
* -p 8200:8200: Exponer el puerto 8200 del contenedor.
* -e 'VAULT\_ADDR=http://127.0.0.1:8200': Establecer variable de entorno VAULT\_ADDR
* -e 'VAULT\_API\_ADDR=http://127.0.0.1:8200': Establecer variable de entorno VAULT\_API\_ADDR que se usa para anunciar a otros servidores en el cluster.
* -e 'VAULT\_LOCAL\_CONFIG=…: Se usa para establecer una variable con la configuración de Vault.
  + Listener.tcp.address: Se usa para establecer como escuchará Vault.
  + Listener.tcp.disable: Se usa para deshabilitar el HTTPS.
  + Backend.file.path: Usado para establecer la ruta de almacenamiento de los secretos.
  + default\_lease\_ttl:
  + max\_lease\_ttl:
  + ui: Usado para habilitar la interfaz gráfica.

## Consideraciones importantes para producción:

* Configurar la alta disponibilidad – replicas.
* Activar logs.
* Configurar protocolo de trasporte seguro – https.

### Configurar la alta disponibilidad – replicas:

Para configurar una alta disponibilidad en Vault se deben tener en cuenta ciertas consideraciones:

* Se debe establecer una base de datos que permita la alta disponibilidad, para mayor información: <https://www.vaultproject.io/docs/configuration/storage/index.html>
* Se debe configurar Vault para la alta disponibilidad, estableciendo correctamente los valores de las variables VAULT\_API\_ADDR, VAULT\_ADDR, VAULT\_CLUSTER\_ADDR, para mayor información: <https://www.vaultproject.io/docs/concepts/ha.html>

#### Inconvenientes al implementar alta disponibilidad:

* Al implementar la alta disponibilidad, Vault trabaja con una arquitectura en la cual se tiene una réplica de modo “**Activo**” y el resto de réplicas en modo de “**Espera**”, por lo cual en un cluster de Kubernetes No se pueden solamente crear las réplicas sin configurar y esperar que el balanceador de carga resuelva la alta disponibilidad.

Lo que sucede es que para utilizar Vault se necesita que la sesión actual se conserve y además Vault necesita saber cual es el nodo activo y cuales están en espera, para evitar bloqueos y asegurar un buen funcionamiento.

Para solucionar esto es necesario configurar correctamente la alta disponibilidad de Vault.

* Solucionando el punto anterior, aun se tiene oro problema, y es que si se cuenta con un nodo A y un nodo B de Vault, siendo el nodo A el activo, si el nodo A se cae por alguna razón, Kubernetes lo va a volver a levantar, pero Vault inmediatamente bloqueará dicho nodo impidiendo el acceso por parte de los usuarios, y debido a que este es le nodo activo, no se mandarán peticiones al nodo B sin importar que esté desbloqueado y listo para recibir peticiones.

Para solucionar esto se debe utilizar un balanceador de carga que verifique qué nodo está activo y desbloqueado para poderle mandar peticiones en caso de que otro falle. Un servicio de Kubernetes como es el “LoadBalancer” no cuenta con dichas configuraciones, éste solo distribuye la carga sin validar al respecto. Por esto es que se ha procedido a configurar un balanceador de carga externo como lo es **HAProxy** en kubernetes.

HAProxy se debe configurar de manera que haga verificaciones de salud de los nodos de Vault cada cierto tiempo, por parte de HAProxy esto se logra haciendo peticiones al servicio **/v1/sys/health** de cada Vault.

El servicio **/v1/sys/health** de Vault devuelve un código HTTP según las condiciones en que se encuentre.

200 si está inicializado, sin sellar y activo

429 si no está sellado y en espera

472 si la replicación del modo de recuperación de datos es secundaria y activa

473 si el rendimiento en espera

501 si no se inicializa

503 si está sellado

Para mayor información:

<https://www.vaultproject.io/api/system/health.html>

## Vault en Openshift

### Inconvenientes

* Al configurar la variable de entorno VAULT\_LOCAL\_CONFIG, una vez inicie Vault, creará un archivo de configuración en la ruta /vault/config/local.json el cual necesita de que la ejecución de Vault se esté realizando desde un usuario con permisos de escritura root, lo cual Openshift no tiene permitido.
* Para que Vault pueda realizar el bloqueo de memoria e impedir que los datos sean intercambiados en el disco, se necesita que la ejecución de Vault se esté realizando desde un usuario con permisos de escritura root, lo cual Openshift no tiene permitido.