bitwarden | Article du Centre d'Aide

SELF-HOSTING > INSTALLER & DÉPLOYER DES GUIDES >

Déploiement Azure AKS

Afficher dans le centre d'aide:
https://bitwarden.com/help/azure-aks-deployment/

Déploiement Azure AKS

Cet article explore comment vous pourriez modifier votre déploiement de Bitwarden auto-hébergé Helm Chart en fonction des offres spécifiques d'Azure et AKS.

Contrôleurs d'entrée

nginx

Un contrôleur d'ingress nginx est défini par défaut dans `my-values.yaml`. Si vous utilisez cette option :

1. Créez un contrôleur d'ingress nginx de base.
2. Décommentez les valeurs dans la section `general.ingress.annotations:` de `my-values.yaml` et personnalisez-les selon vos besoins.

Passerelle d'Application Azure

Les clients Azure peuvent toutefois préférer utiliser une passerelle d'application Azure comme contrôleur d'entrée pour leur cluster AKS.

Avant d'installer le graphique

Si vous préférez cette option, **avant** d'installer le graphique, vous devez :

1. Activez le contrôleur d'entrée Azure Application Gateway pour votre cluster.
2. Mettez à jour votre fichier my-values.yaml, spécifiquement `general.ingress.className:`, `general.ingress.annotations:`, et `general.ingress.paths:` :

Bash

```
general:
  domain: "replaceme.com"
  ingress:
    enabled: true
    className: "azure-application-gateway" # This value might be different depending on how yo
u created your ingress controller.  Use "kubectl get ingressclasses -A" to find the name if unsu
re.
     ## - Annotations to add to the Ingress resource.
    annotations:
      appgw.ingress.kubernetes.io/ssl-redirect: "true"
      appgw.ingress.kubernetes.io/use-private-ip: "false" # This might be true depending on your
setup.
      appgw.ingress.kubernetes.io/rewrite-rule-set: "bitwarden-ingress" # Make note of whatever
you set this value to.  It will be used later.
      appgw.ingress.kubernetes.io/connection-draining: "true" # Update as necessary.
      appgw.ingress.kubernetes.io/connection-draining-timeout: "30" # Update as necessary.
    ## - Labels to add to the Ingress resource.
    labels: {}
    # Certificate options.
    tls:
      # TLS certificate secret name.
      name: tls-secret
      # Cluster cert issuer (e.g. Let's Encrypt) name if one exists.
      clusterIssuer: letsencrypt-staging
    paths:
      web:
        path: /(.*)
        pathType: ImplementationSpecific
      attachments:
        path: /attachments/(.*)
        pathType: ImplementationSpecific
      api:
        path: /api/(.*)
        pathType: ImplementationSpecific
      icons:
```

```
      path: /icons/(.*)
      pathType: ImplementationSpecific
    notifications:
      path: /notifications/(.*)
      pathType: ImplementationSpecific
    events:
      path: /events/(.*)
      pathType: ImplementationSpecific
    scim:
       path: /scim/(.*)
       pathType: ImplementationSpecific
    sso:
      path: /(sso/.*)
      pathType: ImplementationSpecific
    identity:
      path: /(identity/.*)
      pathType: ImplementationSpecific
    admin:
      path: /(admin/?.*)
      pathType: ImplementationSpecific
```

3. Si vous allez utiliser l'exemple fourni par Let's Encrypt pour votre certificat TLS, mettez à jour `spec.acme.solvers.ingress.class:` dans le script lié ici à `"azure/application-gateway"`.

4. Dans le Portail Azure, créez un ensemble de réécriture vide pour le Gateway d'Application :

    1. Naviguez vers **Équilibrage de charge** > **Passerelle d'application** dans le portail Azure et sélectionnez votre passerelle d'application.

    2. Sélectionnez l'onglet **Réécritures** .

    3. Sélectionnez le bouton ┼ **Réécrire le paramètre**.

    4. Définissez la **Nom** à la valeur spécifiée pour `appgw.ingress.kubernetes.io/rewrite-rule-set:` dans `my-values.yaml`, dans cet exemple `bitwarden-ingress`.

    5. Sélectionnez **Suivant** et **Créer**.

Après avoir installé le graphique

Après l'installation du graphique, vous devrez également créer des règles pour votre ensemble de réécriture :

1. Rouvrez l'ensemble de réécriture vide que vous avez créé avant d'installer le graphique.

2. Sélectionnez tous les chemins de routage qui commencent par `pr-bitwarden-auto-héberg-ingress...`, désélectionnez ceux qui ne commencent pas par ce préfixe, et sélectionnez **Suivant**.

3. Sélectionnez le bouton ┼ **Ajouter une règle de réécriture**. Vous pouvez donner à votre règle de réécriture n'importe quel nom et n'importe quelle séquence.

4. Ajoutez la condition suivante :

   - **Type de variable à vérifier**: Variable du serveur
   - **Variable du serveur**: chemin_uri
   - **Sensible à la casse** : Non
   - **Opérateur** : égal (=)
   - **Modèle à correspondre**: `^(\/(?!admin)(?!identité)(?!sso)[^\/]*)\/(.*)`

5. Ajoutez l'action suivante :

   - **Type de réécriture** : URL
   - **Type d'action**: Définir
   - **Composants**: Chemin d'URL
   - **Valeur du chemin URL**: `/{var_uri_path_2}`
   - **Réévaluer la carte du chemin**: Non vérifié

6. Sélectionnez **Créer**.

Création d'une classe de stockage

Le déploiement nécessite une classe de stockage partagé que vous fournissez, qui doit supporter ReadWriteMany. L'exemple suivant est un script que vous pouvez exécuter dans l'Azure Cloud Shell pour créer une classe de stockage de fichiers Azure qui répond à l'exigence :

> **⚠ Warning**
>
> L'exemple suivant est illustratif, assurez-vous d'attribuer les autorisations en fonction de vos propres exigences de sécurité.

```Bash
cat <<EOF | kubectl apply -n bitwarden -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
  namespace: bitwarden
provisioner: file.csi.azure.com
allowVolumeExpansion: true
mountOptions:
  - dir_mode=0777
  - file_mode=0777
  - uid=0
  - gid=0
  - mfsymlinks
  - cache=strict
  - actimeo=30
parameters:
  skuName: Standard_LRS
EOF
```

Vous devez définir la valeur `sharedStorageClassName` dans `my-values.yaml` à n'importe quel nom que vous donnez à la classe, dans cet exemple :

```Bash
sharedStorageClassName: "azure-file"
```

Utilisation du pilote CSI Azure Key Coffre

Le déploiement nécessite des objets secrets Kubernetes pour définir des valeurs sensibles pour votre déploiement. Bien que la commande `kubectl create secret` puisse être utilisée pour définir des secrets, les clients Azure peuvent préférer utiliser Azure Key Vault et le pilote Secrets Store CSI pour AKS :

> **Tip**
> Ces instructions supposent que vous avez déjà configuré un coffre de clés Azure. Sinon, créez-en un maintenant.

1. Ajoutez le support du pilote CSI du magasin de secrets à votre cluster avec la commande suivante :

```bash
az aks enable-addons --addons azure-keyvault-secrets-provider --name myAKSCluster --resource-gro
up myResourceGroup
```

L'extension crée une identité gérée assignée par l'utilisateur que vous pouvez utiliser pour vous authentifier à votre coffre de clés, cependant vous avez d'autres options pour le contrôle d'accès d'Identité. Si vous utilisez l'identité gérée assignée par l'utilisateur créée, vous devrez explicitement lui attribuer **Secret** > **Obtenir** l'accès à celle-ci (apprenez comment).

2. Créez une SecretProviderClass, comme dans l'exemple suivant. Notez que cet exemple contient des espaces réservés que vous devez remplacer et diffère selon que vous utilisez le pod SQL inclus ou votre propre serveur SQL :

Bash

```
cat <<EOF | kubectl apply -n bitwarden -f -
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: bitwarden-azure-keyvault-csi
  labels:
    app.kubernetes.io/component: secrets
  annotations:
spec:
  provider: azure
  parameters:
    useVMManagedIdentity: "true" # Set to false for workload identity
    userAssignedIdentityID: "<REPLACE>" # Set the clientID of the user-assigned managed identity
to use
    # clientID: "<REPLACE>" # Setting this to use workload identity
    keyvaultName: "<REPLACE>"
    cloudName: "AzurePublicCloud"
    objects: |
      array:
        - |
          objectName: installationid
          objectAlias: installationid
          objectType: secret
          objectVersion: ""
        - |
          objectName: installationkey
          objectAlias: installationkey
          objectType: secret
          objectVersion: ""
        - |
          objectName: smtpusername
          objectAlias: smtpusername
          objectType: secret
          objectVersion: ""
        - |
```

```
        objectName: smtppassword
        objectAlias: smtppassword
        objectType: secret
        objectVersion: ""
      - |
        objectName: yubicoclientid
        objectAlias: yubicoclientid
        objectType: secret
        objectVersion: ""
      - |
        objectName: yubicokey
        objectAlias: yubicokey
        objectType: secret
        objectVersion: ""
      - |
        objectName: hibpapikey
        objectAlias: hibpapikey
        objectType: secret
        objectVersion: ""
      - |
        objectName: sapassword #-OR- dbconnectionstring if external SQL
        objectAlias: sapassword #-OR- dbconnectionstring if external SQL
        objectType: secret
        objectVersion: ""
  tenantId: "<REPLACE>"
secretObjects:
- secretName: "bitwarden-secret"
  type: Opaque
  data:
  - objectName: installationid
    key: globalSettings__installation__id
  - objectName: installationkey
    key: globalSettings__installation__key
    key: globalSettings__mail__smtp__username
  - objectName: smtppassword
    key: globalSettings__mail__smtp__password
  - objectName: yubicoclientid
```

```
      key: globalSettings__yubico__clientId
    - objectName: yubicokey
      key: globalSettings__yubico__key
    - objectName: hibpapikey
       key: globalSettings__hibpApiKey
    - objectName: sapassword #-OR- dbconnectionstring if external SQL
      key: SA_PASSWORD #-OR- globalSettings__sqlServer__connectionString if external SQL
EOF
```

3. Utilisez les commandes suivantes pour définir les valeurs de secrets requises dans le coffre de clés :

> ⚠ **Warning**
> Cet exemple enregistrera des commandes dans l'historique de votre shell. D'autres méthodes peuvent être envisagées pour définir un secret de manière sécurisée dans les paramètres.

Bash

```
kvname=<REPLACE>
az keyvault secret set --name installationid --vault-name $kvname --value <REPLACE>
az keyvault secret set --name installationkey --vault-name $kvname --value <REPLACE>
az keyvault secret set --name smtpusername --vault-name $kvname --value <REPLACE>
az keyvault secret set --name smtppassword --vault-name $kvname --value <REPLACE>
az keyvault secret set --name yubicoclientid --vault-name $kvname --value <REPLACE>
az keyvault secret set --name yubicokey --vault-name $kvname --value <REPLACE>
az keyvault secret set --name hibpapikey --vault-name $kvname --value <REPLACE>
az keyvault secret set --name sapassword --vault-name $kvname --value <REPLACE>
# - OR -
# az keyvault secret set --name dbconnectionstring --vault-name $kvname --value <REPLACE>
```

4. Dans votre fichier `my-values.yaml`, définissez les valeurs suivantes :

- `secrets.secretName`: Définissez cette valeur sur le `secretName` défini dans votre SecretProviderClass.
- `secrets.secretProviderClass`: Définissez cette valeur sur le `metadata.name` défini dans vos paramètres de SecretProviderClass.