bitwarden | Help Center Article

SELF-HOSTING > INSTALL & DEPLOY GUIDES >

Azure AKS Deployment

View in the help center:
https://bitwarden.com/help/azure-aks-deployment/

Azure AKS Deployment

This article dives into how you might alter your Bitwarden self-hosted Helm Chart deployment based on the specific offerings of Azure and AKS.

Ingress controllers

nginx

An nginx ingress controller is defined by default in `my-values.yaml`. If you use this option:

1. Create a basic nginx ingress controller.
2. Uncomment the values in the `general.ingress.annotations:` section of `my-values.yaml` and customize them as needed.

Azure Application Gateway

Azure customers may, however, prefer to use an Azure Application Gateway as the ingress controller for their AKS cluster.

Before installing the chart

If you prefer this option, **before** installing the chart you must:

1. Enable the Azure Application Gateway ingress controller for your cluster.
2. Update your my-values.yaml file, specifically `general.ingress.className:`, `general.ingress.annotations:`, and `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. If you're going to use the provided Let's Encrypt example for your TLS certificate, update `spec.acme.solvers.ingress.class:` in the script linked here to `"azure/application-gateway"`.

4. In the Azure Portal, create an empty rewrite set for Application Gateway:

    1. Navigate to the **Load balancing** > **Application Gateway** in the Azure Portal and select your Application Gateway.

    2. Select the **Rewrites** blade.

    3. Select the + **Rewrite set** button.

    4. Set the **Name** to the value specified for `appgw.ingress.kubernetes.io/rewrite-rule-set:` in `my-values.yaml`, in this example `bitwarden-ingress`.

    5. Select **Next** and **Create**.

After installing the chart

After installing the chart, you will also be required to create rules for your rewrite set:

1. Re-open the empty rewrite set you created before installing the chart.

2. Select all routing paths that begin with `pr-bitwarden-self-host-ingress...`, de-select any that do not begin with that prefix, and select **Next**.

3. Select the + **Add Rewrite rule** button. You can give your rewrite rule any name and any sequence.

4. Add the following condition:

   - **Type of variable to check**: Server variable
   - **Server variable**: uri_path
   - **Case-sensitive**: No
   - **Operator**: equal (=)
   - **Pattern to match**: `^(\/(?!admin)(?!identity)(?!sso)[^\/]*)\/(.*)`

5. Add the following action:

   - **Rewrite type**: URL
   - **Action type**: Set
   - **Components**: URL path
   - **URL path value**: `/{var_uri_path_2}`
   - **Re-evaluate path map**: Unchecked

6. Select **Create**.

Creating a storage class

Deployment requires a shared storage class that you provide, which must support ReadWriteMany. The following example is a script you can run in the Azure Cloud Shell to create an Azure File Storage class that meets the requirement:

> ⚠ **Warning**
>
> The following is an illustrative example, be sure to assign permissions according to your own security requirements.

```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
```

You must set the `sharedStorageClassName` value in `my-values.yaml` to whatever name you give the class, in this example:

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

Using Azure Key Vault CSI Driver

Deployment requires Kubernetes secrets objects to set sensitive values for your deployment. While the `kubectl create secret` command can be used to set secrets, Azure customers may prefer to use Azure Key Vault and the Secrets Store CSI Driver for AKS:

> **Tip**
>
> These instructions assume you already an have Azure Key Vault setup. If not, create one now.

1. Add Secrets Store CSI Driver support to your cluster with the following command:

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

The add-on creates a user-assigned managed identity you can use to authenticate to your key vault, however you have other options for identity access control. If you use the created user-assigned managed identity, you will need to explicitly assign **Secret** > **Get** access to it (learn how).

2. Create a SecretProviderClass, as in the following example.

   The `parameters` section of the following YAML file is accurate for most environments. However, depending on your setup, you may need to change some values; for example, `cloudName` should be set to `AzureUSGovernmentCloud` for Azure US Government Cloud. Consult Microsoft's documentation for full details.

   The `parameters` section also contains `<REPLACE>` placeholders that you must replace, and will be slightly different depending on if you are using the included SQL pod or using your own SQL server.

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. Use the following commands to set the required secrets values in Key Vault:

> ⚠ **Warning**
> This example will record commands to your shell history. Other methods may be considered to securely set a secret.

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. In your `my-values.yaml` file, set the following values:

- `secrets.secretName`: Set this value to the `secretName` defined in your SecretProviderClass.
- `secrets.secretProviderClass`: Set this value to the `metadata.name` defined in your SecretProviderClass.