bitwarden | Help Center Article

SELF-HOSTING > INSTALL & DEPLOY GUIDES >

AWS EKS Deployment

View in the help center:
https://bitwarden.com/help/aws-eks-deployment/

AWS EKS Deployment

This article dives into how you might alter your Bitwarden self-hosted Helm Chart deployment based on the specific offerings of AWS and Elastic Kubernetes Service (EKS).

Note that certain add-ons documented in this article will require that your EKS cluster has at least one node already launched.

Ingress controller

An nginx controller is defined by default in `my-values.yaml`, and will require an AWS Network Load Balancer. AWS Application Load Balancers (ALB) are not currently recommended as they do not support path rewrites and path-based routing.

Note

The following assumes that you have an SSL certificate saved in AWS Certificate Manager, as you will need a certificate Amazon Resource Name (ARN).

You also must have at least 1 node already running in your cluster.

To connect a Network Load Balancer to your cluster:

1. Follow these instructions to create an IAM policy and role, and to install the AWS Load Balancer Controller in your cluster.

2. Run the following commands to setup an ingress controller for your cluster. This will create an AWS Network Load Balancer. Note that there are values you **must** replace as well as values you can configure to suit your needs in this example command:

Bash

```bash
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm upgrade ingress-nginx ingress-nginx/ingress-nginx -i \
  --namespace kube-system \
  --set-string controller.service.annotations.'service\.beta\.kubernetes\.io/aws-load-balancer-backend-protocol'="ssl" \
  --set-string controller.service.annotations.'service\.beta\.kubernetes\.io/aws-load-balancer-cross-zone-load-balancing-enabled'="true" \
  --set-string controller.service.annotations.'service\.beta\.kubernetes\.io/aws-load-balancer-type'="external" \
  --set-string controller.service.annotations.'service\.beta\.kubernetes\.io/aws-load-balancer-nlb-target-type'="instance" \
  --set-string controller.service.annotations.'service\.beta\.kubernetes\.io/aws-load-balancer-scheme'="internet-facing" \
  --set-string controller.service.annotations.'service\.beta\.kubernetes\.io/aws-load-balancer-ssl-cert'="arn:aws:acm:REPLACEME:REPLACEME:certificate/REPLACEME" \ #Replace with the ARN for your certificate
  --set-string controller.service.annotations.'service\.beta\.kubernetes\.io/aws-load-balancer-ssl-ports'="443" \
  --set controller.service.externalTrafficPolicy="Local"
```

3. Update your `my-values.yaml` file according to the following example, making sure to replace any `REPLACE` placeholders:

Bash

```
general:
  domain: "REPLACEME.com"
  ingress:
    enabled: true
    className: "nginx"
     ## - Annotations to add to the Ingress resource
    annotations:
      nginx.ingress.kubernetes.io/ssl-redirect: "true"
      nginx.ingress.kubernetes.io/use-regex: "true"
      nginx.ingress.kubernetes.io/rewrite-target: /$1
    ## - Labels to add to the Ingress resource
    labels: {}
    # Certificate options
    tls:
      # TLS certificate secret name
      name: # Handled via the NLB defined in the ingress controller
      # Cluster cert issuer (ex. Let's Encrypt) name if one exists
      clusterIssuer:
    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
```

Create a storage class

Deployment requires a shared storage class that you provide, which must support ReadWriteMany. The following example of how to create a storage class that meets the requirement:

Tip

The following assumes that you have an AWS Elastic File System (EFS) created. If you don't create one now. In either case, take note of your EFS' **File system ID** as you will need it during this process.

1. Get the Amazon EFS CSI driver add-on for your EKS cluster. This will require that you create an OIDC provider for your cluster and create an IAM role for the driver.

2. In the AWS CloudShell, replace the `file_system_id= "REPLACE"` variable in the the following script and run it in the AWS CloudShell:

   Warning

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

```bash
file_system_id="REPLACE"

cat << EOF | kubectl apply -n bitwarden -f -
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: shared-storage
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap
  fileSystemId: $file_system_id
  directoryPerms: "777" # Change for your use case
  uid: "2000" # Change for your use case
  gid: "2000" # Change for your use case
  basePath: "/dyn1"
  subPathPattern: "\${.PVC.name}"
  ensureUniqueDirectory: "false"
  reuseAccessPoint: "false"
mountOptions:
  - iam
  - tls
EOF
```

3. Set the `sharedStorageClassName` value in `my-values.yaml` to whatever name you give the class in `metadata.name:`, in this example:

```bash
sharedStorageClassName: "shared-storage"
```

Using AWS Secrets Manager

Deployment requires Kubernetes secrets objects to set sensitive values for your deployment. While the `kubectl create secret` command can be used to set secrets, AWS customers may prefer to use AWS Secrets Manager and the AWS Secrets and Configuration Provider (ACSP) for Kubernetes Secrets Store CSI Driver.

You will need the following secrets stored in AWS Secrets Manager. Note that you can change the **Keys** used here but must also make changes to subsequent steps if you do:

Key	Value
`installationid`	A valid installation id retrieved from https://bitwarden.com/host. For more information, see what are my installation id and installation key used for?
`installationkey`	A valid installation key retrieved from https://bitwarden.com/host. For more information, see what are my installation id and installation key used for?
`smtpusername`	A valid username for your SMTP server.
`smtppassword`	A valid password for the entered SMTP server username.
`yubicoclientid`	Client ID for YubiCloud Validation Service or self-hosted Yubico Validation Server. If YubiCloud, get your client ID and secret key here.
`yubicokey`	Secret key for YubiCloud Validation Service or self-hosted Yubico Validation Server. If YubiCloud, get your client ID and secret key here.
globalSettings__hibpApiKey	Your HaveIBeenPwned (HIBP) API Key, available here. This key allows users to run the Data Breach report and to check their master password for presence in breaches when they create an account.
If you're using the Bitwarden SQL pod, `sapassword`. If you're using your own SQL server, `db connectionString.`	Credentials for the database connected to your Bitwarden instance. What is required will depend on whether you're using the included SQL pod or an external SQL server.

1. Once your secrets are securely stored, install ACSP.

2. Create a permissions policy to allow access to your secrets. This policy **must** grant `secretsmanager:GetSecretValue` and `secretsmanager:DescribeSecret` permission, for example:

```bash
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "secretsmanager:DescribeSecret",
      "secretsmanager:GetSecretValue"
    ],
    "Resource": "arn:aws:secretsmanager:REPLACEME:REPLACEME:secret:REPLACEME"
  }
}
```

3. Create a service account that has access to your secrets via the created permissions policy, for example:

```bash
CLUSTER_NAME="REPLACE"
ACCOUNT_ID="REPLACE" # replace with your AWS account ID
ROLE_NAME="REPLACE" # name of a role that will be created in IAM
POLICY_NAME="REPLACE" # the name of the policy you created earlier
eksctl create iamserviceaccount \
  --cluster=$CLUSTER_NAME \
  --namespace=bitwarden \
  --name=bitwarden-sa \
  --role-name $ROLE_NAME \
  --attach-policy-arn=arn:aws:iam::$ACCOUNT_ID:policy/$POLICY_NAME \
  --approve
```

4. Next, create a SecretProviderClass, as in the following example. Be sure to replace the `region` with your region and the `objectName` with the name of the Secrets Manager secret you created (**Step 1**):

Bash

```bash
cat <<EOF | kubectl apply -n bitwarden -f -
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: bitwarden-secrets-manager-csi
  labels:
    app.kubernetes.io/component: secrets
  annotations:
spec:
  provider: aws
  parameters:
    region: REPLACE
    objects: |
      - objectName: "REPLACE"
        objectType: "secretsmanager"
        objectVersionLabel: "AWSCURRENT"
        jmesPath:
          - path: installationid
            objectAlias: installationid
          - path: installationkey
            objectAlias: installationkey
          - path: smtpusername
            objectAlias: smtpusername
          - path: smtppassword
            objectAlias: smtppassword
          - path: yubicoclientid
            objectAlias: yubicoclientid
          - path: yubicokey
            objectAlias: yubicokey
          - path: hibpapikey
            objectAlias: hibpapikey
          - path: sapassword #-OR- dbconnectionstring if external SQL
            objectAlias: sapassword #-OR- dbconnectionstring if external SQL
  secretObjects:
  - secretName: "bitwarden-secret"
```

```
    type: Opaque
    data:
    - objectName: installationid
      key: globalSettings__installation__id
    - objectName: installationkey
      key: globalSettings__installation__key
    - objectName: smtpusername
      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
```

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

- `secrets.secretName`: Set to the `secretName` defined in your SecretProviderClass (**Step 3**).
- `secrets.secretProviderClass`: Set to the `metedata.name` defined in your SecretProviderClass (**Step 3**).
- `component.admin.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.api.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.attachments.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.events.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.icons.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.identity.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.notifications.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.scim.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.sso.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `component.web.podServiceAccount`: Set to the name defined for your service account (**Step 2**).

- `database.podServiceAccount`: Set to the name defined for your service account (**Step 2**).
- `serviceAccount.name`: Set to the name defined for your service account (**Step 2**).
- `serviceAccount.deployRolesOnly`: Set to `true`.