bitwarden | Help Center Article

SELF-HOSTING > INSTALL & DEPLOY GUIDES >

# Self-host with Helm

View in the help center:
https://bitwarden.com/help/self-host-with-helm/

# Self-host with Helm

This article will walk you through the procedure to install and deploy Bitwarden in different Kubernetes deployments using a Helm chart.

This article will describe the generic steps for hosting Bitwarden on Kubernetes. Provider-specific guides are available to dive into how you might alter a deployment based on each provider's specific offerings:

- Azure AKS Deployment
- OpenShift Deployment
- AWS EKS Deployment

## Requirements

Before proceeding with the installation, ensure the following requirements are met:

- kubectl is installed.
- Helm 3 is installed.
- You have an SSL certificate and key or access to creating one via a certificate provider.
- You have a SMTP server or access to a cloud SMTP provider.
- A storage class that supports ReadWriteMany.
- You have an installation id and key retrieved from https://bitwarden.com/host.

## Prepare the chart

### Add the repo to Helm

Add the repo to Helm using the following commands:

```bash
helm repo add bitwarden https://charts.bitwarden.com/
helm repo update
```

### Create a namespace

Create a namespace to deploy Bitwarden to. Our documentation assumes a namespace called `bitwarden`, so be sure to modify commands if you choose a different name.

```bash
kubectl create namespace bitwarden
```

## Create a configuration

Create a `my-values.yaml` configuration file, which you will use to customize your deployment, using the following command:

```bash
helm show values bitwarden/self-host > my-values.yaml
```

At a minimum, you must configure the following values in your `my-values.yaml` file:

| Value | Description |
|---|---|
| `general.domain:` | The domain that will point to your cluster's public IP address. |
| `general.ingress.enabled:` | Whether to use the nginx ingress controller defined in the chart (see an example using a non-included ingress controller). |
| `general.ingress.className:` | For example, `"nginx"` or `"azure-application-gateway"` (see an example). Set `general.ingress.enabled: false` to use other ingress controllers. |
| `general.ingress.annotations:` | Annotations to add to the ingress controller. If you're using the included nginx controller, defaults are provided that you must uncomment and can customize as needed. |
| `general.ingress.paths:` | If you're using the default nginx controller, defaults are provided that you can customize as needed. |
| `general.ingress.cert.tls.name:` | The name of your TLS certificate. We will walk through an example later, so enter it now if you have it or circle back later. |
| `general.ingress.cert.tls.clusterIssuer:` | The name of your TLS certificate issuer. We will walk through an example later, so enter it now if you have it or circle back later. |
| `general.email.replyToEmail:` | Email address used for invitations, typically `no_reply@smtp_host`. |

| Value | Description |
| --- | --- |
| `general.email.smtpHost:` | Your SMTP server hostname or IP address. |
| `general.email.smtpPort:` | The SMTP port used by the SMTP server. |
| `general.email.smtpSsl:` | Whether your SMTP server uses an encryption protocol (`true` = SSL, `false` = TLS). |
| `enableCloudCommunication:` | Set to `true` to allow communication between your server and our cloud system. Doing so enables billing and license sync. |
| `cloudRegion:` | By default, US. Set to EU if your organization was started via the EU cloud server. |
| `sharedStorageClassName:` | The name of the shared storage class, which you will need to provide and must support ReadWriteMany (see an example using Azure File Storage) unless it's a single-node cluster. |
| `secrets.secretName:` | The name of your Kubernetes secret object. You will create this object in the next step, so decide on a name now or circle back to this value. |
| `database.enabled:` | Whether to use the SQL pod included in the chart. Only set to `false` if you're using an external SQL server. |
| `component.scim.enabled` | The SCIM pod is disabled by default. To enable the SCIM pod, set value `= true`. |
| `volume.logs.enabled:` | While not required, we recommend setting to `true` for troubleshooting purposes. |

## Create a secret object

Create a Kubernetes secret object to set, at a minimum, the following values:

| Value | Description |
| --- | --- |
| `globalSettings__installation__id` | A valid installation id retrieved from https://bitwarden.com/host. For more information, see what are my installation id and installation key used for? |
| `globalSettings__installation__key` | A valid installation key retrieved from https://bitwarden.com/host. For more information, see what are my installation id and installation key used for? |
| `globalSettings__mail__smtp__username` | A valid username for your SMTP server. |
| `globalSettings__mail__smtp__password` | A valid password for the entered SMTP server username. |
| `globalSettings__yubico__clientId` | Client ID for YubiCloud Validation Service or self-hosted Yubico Validation Server. If YubiCloud, get your client ID and secret key here. |
| `globalSettings__yubico__key` | 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, `SA_PASSWORD`<br><br>If you're using your own SQL server, `globalSettings__sqlServer__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. |

For example, using the `kubectl create secret` command to set these values would look like the following:

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

```bash
kubectl create secret generic custom-secret -n bitwarden \
    --from-literal=globalSettings__installation__id="REPLACE" \
    --from-literal=globalSettings__installation__key="REPLACE" \
    --from-literal=globalSettings__mail__smtp__username="REPLACE" \
    --from-literal=globalSettings__mail__smtp__password="REPLACE" \
    --from-literal=globalSettings__yubico__clientId="REPLACE" \
    --from-literal=globalSettings__yubico__key="REPLACE" \
    --from-literal=globalSettings__hibpApiKey="REPLACE" \
    --from-literal=SA_PASSWORD="REPLACE"
```

Don't forget to set the `secrets.secretName:` value in `my-values.yaml` to the name of the created secret, in this case `custom-secret`.

### Example certificate setup

Deployment requires a TLS certificate and key, or access to a creating one via certificate provider. The following example will walk you through using cert-manager to generate a certificate with Let's Encrypt:

1. Install cert-manager on the cluster using the following command:

   ```bash
   kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.11.0/cert-manager.yaml
   ```

2. Define a certificate issuer. Bitwarden recommends using the **Staging** configuration in this example until your DNS records have been pointed to your cluster. Be sure to replace the `email:` placeholder with a valid value:

### ⇒Staging

```Bash
cat <<EOF | kubectl apply -n bitwarden -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    email: me@example.com
    privateKeySecretRef:
      name: tls-secret
    solvers:
      - http01:
          ingress:
            class: nginx #use "azure/application-gateway" for Application Gateway ingress
EOF
```

### ⇒Production

```
Bash

cat <<EOF | kubectl apply -n bitwarden -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: me@example.com
    privateKeySecretRef:
      name: tls-secret
    solvers:
      - http01:
          ingress:
            class: nginx #use "azure/application-gateway" for Application Gateway ingress
EOF
```

3. If you haven't already, be sure to set the `general.ingress.cert.tls.name:` and `general.ingress.cert.tls.clusterIssuer:` values in `my-values.yaml`. In this example, you would set:
   - `general.ingress.cert.tls.name: tls-secret`
   - `general.ingress.cert.tls.clusterIssuer: letsencrypt-staging`

## Adding rawManifest files

The Bitwarden self-host Helm Chart allows you to include other Kubernetes manifest files either pre- or post-install. To do this, update the `rawManifests` section of the chart (learn more). This is useful, for example, in scenarios where you want to use an ingress controller other than the nginx controller defined by default.

## Install the chart

To install Bitwarden with the configuration setup in `my-values.yaml`, run the following command:

```
Bash

helm upgrade bitwarden bitwarden/self-host --install --namespace bitwarden --values my-values.yaml
```

Congratulations! Bitwarden is now up and running at `https://your.domain.com`, as defined in `my-values.yaml`. Visit the web vault in your web browser to confirm that it's working. You may now register a new account and log in.

You will need to have setup an SMTP configuration and related secrets in order to verify the email for your new account.

## Next steps

### Database backup and restore

In this repository, we have provided two illustrative example jobs for backing up and restoring the database in the Bitwarden database pod. If you are using your own SQL Server instance that is not deployed as part of this Helm chart, please follow your corporate backup and restore policies.

Database backups and backup policies are ultimately up to the implementor. The backup could be scheduled outside of the cluster to run at a regular interval, or it could be modified to create a CronJob object within Kubernetes for scheduling purposes.

The backup job will create timestamped versions of the previous backups. The current backup is simply called `vault.bak`. These files are placed in the MS SQL backups persistent volume. The restore job will look for `vault.bak` in the same persistent volume.