Déploiement AWS EKS

Déploiement AWS EKS

Cet article se penche sur la manière dont vous pourriez modifier votre déploiement de Bitwarden Helm Chart auto-hébergé en fonction des offres spécifiques d'AWS et du service Elastic Kubernetes (EKS).

Notez que certains modules complémentaires documentés dans cet article nécessiteront que votre cluster EKS ait déjà lancé au moins un nœud.

Contrôleur d'entrée

Un contrôleur nginx est défini par défaut dans `my-values.yaml`, et nécessitera un équilibreur de charge réseau AWS. Les équilibreurs de charge d'application AWS (ALB) ne sont pas actuellement recommandés car ils ne prennent pas en charge la réécriture de chemin et le routage basé sur le chemin.

> **Note**
>
> L'hypothèse suivante suppose que vous avez un certificat SSL enregistré dans AWS Certificate Manager, car vous aurez besoin d'un nom de ressource Amazon (ARN).
>
> Vous devez également avoir au moins 1 nœud déjà en fonctionnement dans votre grappe.

Pour connecter un équilibreur de charge réseau à votre cluster :

1. Suivez ces instructions pour créer une politique de sécurité IAM et un rôle, et pour installer le contrôleur de charge AWS dans votre cluster.

2. Exécutez les commandes suivantes pour configurer un contrôleur d'ingress pour votre cluster. Cela créera un équilibreur de charge réseau AWS. Notez qu'il y a des valeurs que vous **devez** remplacer ainsi que des valeurs que vous pouvez configurer selon vos besoins dans cette commande exemple :

```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. Mettez à jour votre fichier `my-values.yaml` selon l'exemple suivant, en veillant à remplacer tous les espaces réservés `REPLACE` :

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
```

Créez une classe de stockage

Le déploiement nécessite une classe de stockage partagé que vous fournissez, qui doit supporter ReadWriteMany. L'exemple suivant montre comment créer une classe de stockage qui répond à l'exigence :

> **Tip**
>
> L'information suivante suppose que vous avez créé un système de fichiers élastiques AWS (EFS). Si vous n'en créez pas un maintenant. Dans les deux cas, prenez note de votre identifiant de **système de fichiers EFS** car vous en aurez besoin pendant ce processus.

1. Obtenez le module complémentaire de pilote Amazon EFS CSI pour votre cluster EKS. Cela nécessitera que vous créiez un fournisseur OIDC pour votre cluster et créiez un rôle IAM pour le pilote.

2. Dans le CloudShell AWS, remplacez la variable `file_system_id= "REPLACE"` dans le script suivant et exécutez-le dans le CloudShell AWS :

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

```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. Définissez la valeur de `sharedStorageClassName` dans `my-values.yaml` à n'importe quel nom que vous donnez à la classe dans `metadata.name:`, dans cet exemple :

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

En utilisant AWS Secrets Manager

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 AWS peuvent préférer utiliser AWS Secrets Manager et le fournisseur de secrets et de configuration AWS (ACSP) pour le pilote CSI du magasin de secrets Kubernetes.

Vous aurez besoin des secrets suivants stockés dans AWS Secrets Manager. Notez que vous pouvez changer les **Clés** utilisées ici, mais vous devez également apporter des modifications aux étapes suivantes si vous le faites :

Clé	Valeur
`idInstallation`	Un identifiant d'installation valide récupéré depuis https://bitwarden.com/host. Pour plus d'informations, voir à quoi servent mon identifiant d'installation et ma clé d'installation ?
`clé d'installation`	Une clé d'installation valide récupérée depuis https://bitwarden.com/host. Pour plus d'informations, voir à quoi servent mon identifiant d'installation et ma clé d'installation ?
`nomd'utilisateursmtp`	Un nom d'utilisateur valide pour votre serveur SMTP.
`motdepasseSMTP`	Un mot de passe valide pour le nom d'utilisateur du serveur SMTP entré.
`yubicoclientid`	ID du client pour le service de validation YubiCloud ou le serveur de validation Yubico auto-hébergé. Si YubiCloud, obtenez votre ID client et clé secrète ici.
`yubicokey`	Clé secrète pour le service de validation YubiCloud ou le serveur de validation Yubico auto-hébergé. Si YubiCloud, obtenez votre ID client et clé secrète ici.
paramètresGlobaux__hibpCleApi	Votre clé API HavelBeenPwned (HIBP), disponible ici. Cette clé permet aux utilisateurs d'exécuter le rapport de brèche de données et de vérifier la présence de leur mot de passe principal dans les brèches lorsqu'ils créent un compte.
Si vous utilisez le pod SQL de Bitwarden, `sapassword`. Si vous utilisez votre propre serveur SQL, `dbconnectionString`.	Identifiants pour la base de données connectée à votre instance Bitwarden. Ce qui est requis dépendra de si vous utilisez le pod SQL inclus ou un serveur SQL externe.

1. Une fois vos secrets enregistrés en toute sécurité, installez ACSP.

2. Créez une politique d'autorisation pour permettre l'accès à vos secrets. Cette politique **doit** accorder l'`autorisation secretsmanager:GetSecretValue` et l'`autorisation secretsmanager:DescribeSecret`, par exemple :

```
Bash
```

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

3. Créez un compte de service qui a accès à vos secrets via la politique d'autorisation créée, par exemple :

```
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. Ensuite, créez une SecretProviderClass, comme dans l'exemple suivant. Assurez-vous de remplacer la `région` par votre région et le `objectName` par le nom du secret que vous avez créé dans Secrets Manager (**Étape 1**):

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

- `secrets.secretName`: Définissez-le sur le `secretName` défini dans votre SecretProviderClass (**Étape 3**).
- `secrets.secretProviderClass`: Définissez-le sur le `metedata.name` défini dans votre SecretProviderClass (**Étape 3**).
- `component.admin.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.api.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.attachments.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.events.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.icons.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.identity.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.notifications.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.scim.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.sso.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `component.web.podServiceAccount`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).

- `base de données.podServiceCompte`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `serviceAccount.name`: Définissez-le sur le nom défini pour votre compte de service (**Étape 2**).
- `serviceAccount.deployRolesOnly`: Définir à `vrai`.