Red Hat Developer Hub 1.8

Installing Red Hat Developer Hub on Microsoft Azure Kubernetes Service (AKS)

Running Red Hat Developer Hub on Microsoft Azure Kubernetes Service (AKS) by using either the Operator or Helm chart

Last Updated: 2025-11-25

Red Hat Developer Hub 1.8 Installing Red Hat Developer Hub on Microsoft Azure Kubernetes Service (AKS)

Running Red Hat Developer Hub on Microsoft Azure Kubernetes Service (AKS) by using either the Operator or Helm chart

Abstract

Red Hat Developer Hub (RHDH) is an enterprise-grade platform for building developer portals. Administrative users can configure roles, permissions, and other settings to enable other authorized users to deploy a RHDH instance on Microsoft Azure Kubernetes Service (AKS) using either the Operator or Helm chart.

Table of Contents

PREFACE

Red Hat Developer Hub (RHDH) is an enterprise-grade platform for building developer portals. Administrative users can configure roles, permissions, and other settings to enable other authorized users to deploy a RHDH instance on Microsoft Azure Kubernetes Service (AKS) using either the Operator or Helm chart.

CHAPTER 1. INSTALLING DEVELOPER HUB ON MICROSOFT AZURE KUBERNETES SERVICE (AKS) BY USING THE OPERATOR

To benefit from over-the-air updates and catalogs provided by Operator-based applications distributed with the Operator Lifecycle Manager (OLM) framework, consider installing Red Hat Developer Hub by using the Red Hat Developer Hub Operator distributed in the Red Hat Container Registry.

On AKS, the most notable differences over an OpenShift-based installation are:

- The OLM framework and the Red Hat Container Registry are not built-in.
- The Red Hat Container Registry pull-secret is not managed globally.
- To expose the application, Ingresses replace OpenShift Routes.

For clarity, the content is broken down in sections highlighting these platform-specific additional steps.

1.1. INSTALLING THE DEVELOPER HUB OPERATOR ON MICROSOFT AZURE KUBERNETES SERVICE (AKS) BY USING THE OLM FRAMEWORK

The Red Hat Container Registry (registry.redhat.io), based on the Operator Lifecycle Manager (OLM) framework, contains a distribution of the Red Hat Developer Hub Operator, aimed at managing your Red Hat Developer Hub instance lifecycle.

However, on Microsoft Azure Kubernetes Service (AKS):

- The Operator Lifecycle Manager (OLM) framework and the Red Hat Container Registry are not built-in.
- The Red Hat Container Registry pull-secret is not managed globally.

Therefore, install the OLM framework, the Red Hat Container Registry, and provision your Red Hat Container Registry pull secret to install Developer Hub Operator.

Prerequisites

- You have installed the **kubectl** CLI on your local environment .
- Your system meets the sizing requirements for Red Hat Developer Hub .
- You have installed the Operator Lifecycle Manager (OLM) .
- Your credentials to the Red Hat Container Registry :
  - *<redhat_user_name>*
  - *<redhat_password>*
  - *<email>*

Procedure

1. Create the **rhdh-operator** namespace to contain the Red Hat Developer Hub Operator:

```
$ kubectl create namespace rhdh-operator
```

2. Create a pull secret using your Red Hat credentials to pull the container images from the protected Red Hat Container Registry (registry.redhat.io):

```
$ kubectl -n rhdh-operator create secret docker-registry rhdh-pull-secret \
    --docker-server=registry.redhat.io \
    --docker-username=<redhat_user_name> \
    --docker-password=<redhat_password> \
    --docker-email=<email>
```

3. Create a catalog source that contains the Red Hat operators:

```
$ cat <<EOF | kubectl -n rhdh-operator apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: redhat-catalog
spec:
  sourceType: grpc
  image: registry.redhat.io/redhat/redhat-operator-index:v4.19
  secrets:
  - "rhdh-pull-secret"
  displayName: Red Hat Operators
EOF
```

4. Create an operator group to manage your operator subscriptions:

```
$ cat <<EOF | kubectl apply -n rhdh-operator -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: rhdh-operator-group
EOF
```

5. Create a subscription to install the Red Hat Developer Hub Operator:

```
$ cat <<EOF | kubectl apply -n rhdh-operator -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: rhdh
  namespace: rhdh-operator
spec:
  channel: fast
  installPlanApproval: Automatic
  name: rhdh
  source: redhat-catalog
  sourceNamespace: rhdh-operator
  startingCSV: rhdh-operator.v1.8.0
EOF
```

6. To wait until the Operator deployment finishes to be able to run the next step, run:

```
until kubectl -n rhdh-operator get deployment rhdh-operator &>/dev/null; do
  echo -n .
  sleep 3
done
echo "RHDH Operator Deployment created"
```

7. Include your pull secret name in the Operator deployment manifest, to avoid **ImagePullBackOff** errors:

```
$ kubectl -n rhdh-operator patch deployment \
    rhdh-operator --patch '{"spec":{"template":{"spec":{"imagePullSecrets":[{"name":"rhdh-pull-secret"}]}}}}' \
    --type=merge
```

Verification

- Verify the deployment name:

```
$ kubectl get deployment -n rhdh-operator
```

1.2. PROVISIONING YOUR CUSTOM RED HAT DEVELOPER HUB CONFIGURATION

To configure Red Hat Developer Hub, provision your custom Red Hat Developer Hub config maps and secrets to Microsoft Azure Kubernetes Service (AKS) before running Red Hat Developer Hub.

TIP

On Red Hat OpenShift Container Platform, you can skip this step to run Developer Hub with the default config map and secret. Your changes on this configuration might get reverted on Developer Hub restart.

Prerequisites

- By using the Kubernetes CLI ('kubectl'), you have access, with developer permissions, to the Kubernetes cluster aimed at containing your Developer Hub instance.

Procedure

1. For security, store your secrets as environment variables values in an OpenShift Container Platform secret, rather than in clear text in your configuration files. Collect all your secrets in the **secrets.txt** file, with one secret per line in **KEY=value** form.
    - Enter your authentication secrets.
2. Author your custom **app-config.yaml** file. This is the main Developer Hub configuration file. You need a custom **app-config.yaml** file to avoid the Developer Hub installer to revert user edits during upgrades. When your custom **app-config.yaml** file is empty, Developer Hub is using default values.
    a. For a production environment, start with the following setup:

app-config.yaml

```
app:
  title: <Red Hat Developer Hub>
  branding:
    fullLogo: ${BASE64_EMBEDDED_FULL_LOGO}
    fullLogoWidth: 110px
    iconLogo: ${BASE64_EMBEDDED_ICON_LOGO}
backend:
  cache:
    store: redis
    connection: ${REDIS_CONNECTION}
techdocs:
  cache:
    ttl: 3600000
catalog:
  providers:
    <enter_your_provider_configuration>
integrations:
    <enter_your_integrations_configuration>
permission:
  enabled: true
  rbac:
    admin:
      users:
        - name: user:default/<your_policy_administrator_name>
    pluginsWithPermission:
      - catalog
      - scaffolder
      - permission
```

Most fields use environment variables that you defined in secrets in the previous step.

app

title

Enter your Developer Hub instance display name, such as *<Red Hat Developer Hub>*.

branding

Set your custom logo.
Optionally, customize the width of the branding logo by changing value for the **fullLogoWidth** field. The following units are supported: integer, px, em, rem, percentage.

backend

cache

Enable the plugins assets cache.

techdocs

cache

Enable the Techdocs cache.

catalog

provider

Enter your catalog provider configuration.

integrations

Enter your repository discovery configuration.

permissions

Enable Role-based access control. Enter your policy administrator name.

b. Additionally, provision users and enable authentication with your external identity provider .

3. Author your custom **dynamic-plugins.yaml** file to enable plugins. By default, Developer Hub enables a minimal plugin set, and disables plugins that require configuration or secrets, such as the GitHub repository discovery plugin and the Role-based access control (RBAC) plugin. Enable the GitHub repository discovery and the RBAC features:

dynamic.plugins.yaml

```
includes:
  - dynamic-plugins.default.yaml
plugins:
  - package: ./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github
    disabled: false
  - package: ./dynamic-plugins/dist/backstage-community-plugin-rbac
    disabled: false
```

4. Provision your custom configuration files to your AKS cluster.

a. Create the *<my-rhdh-project>* {namespace} aimed at containing your Developer Hub instance.

```
$ oc create namespace my-rhdh-project
```

b. Provision your **app-config.yaml** and **dynamic-plugins.yaml** files respectively to the **my-rhdh-app-config**, and **dynamic-plugins-rhdh** config maps in the *<my-rhdh-project>* project.

```
$ oc create configmap my-rhdh-app-config --from-file=app-config.yaml --
namespace=my-rhdh-project
$ oc create configmap dynamic-plugins-rhdh --from-file=dynamic-plugins.yaml --
namespace=my-rhdh-project
```

Alternatively, create the config maps by using the web console .

c. Provision your **secrets.txt** file to the **my-rhdh-secrets** secret in the *<my-rhdh-project>* project.

```
$ oc create secret generic my-rhdh-secrets --from-file=secrets.txt --namespace=my-
rhdh-project
```

Alternatively, create the secret by using the web console .

Next steps

- Provision your PostgreSQL database secrets
- Provision your dynamic plugins config map
- Provision your RBAC policies config map

1.3. PROVISION YOUR RED HAT CONTAINER REGISTRY PULL SECRET TO YOUR RED HAT DEVELOPER HUB INSTANCE NAMESPACE

On Microsoft Azure Kubernetes Service (AKS), the Red Hat Container Registry pull-secret is not managed globally. Therefore add your pull-secret in your Red Hat Developer Hub instance namespace.

Prerequisites

- Your credentials to the Red Hat Container Registry :
  - *<redhat_user_name>*
  - *<redhat_password>*
  - *<email>*
- You created the **{my-rhdh-project}** namespace on AKS to host your Developer Hub instance.

Procedure

1. Create a pull secret using your Red Hat credentials to pull the container images from the protected Red Hat Container Registry (registry.redhat.io):

   ```
   $ kubectl -n {my-rhdh-namespace} create secret docker-registry my-rhdh-pull-secret \
       --docker-server=registry.redhat.io \
       --docker-username=<redhat_user_name> \
       --docker-password=<redhat_password> \
       --docker-email=<email>
   ```

2. To enable pulling Developer Hub images from the Red Hat Container Registry, add the image pull secret in the default service account within the namespace where the Developer Hub instance is being deployed:

   ```
   $ kubectl patch serviceaccount default \
       -p '{"imagePullSecrets": [{"name": "my-rhdh-pull-secret"}]}' \
       -n {my-rhdh-namespace}
   ```

1.4. USING THE RED HAT DEVELOPER HUB OPERATOR TO RUN DEVELOPER HUB WITH YOUR CUSTOM CONFIGURATION

To use the Developer Hub Operator to run Red Hat Developer Hub with your custom configuration, create your Backstage custom resource (CR) that:

- Mounts files provisioned in your custom config maps.
- Injects environment variables provisioned in your custom secrets.

Prerequisites

- By using the Kubernetes CLI ('kubectl'), you have access, with developer permissions, to the AKS cluster aimed at containing your Developer Hub instance.
- Your administrator has installed the Red Hat Developer Hub Operator in the cluster.
- You have provisioned your custom config maps and secrets in your ***<my-rhdh-project>*** project.
- You have a working default storage class, such as the EBS storage add-on, configured in your EKS cluster.

Procedure

1. Author your Backstage CR in a **my-rhdh-custom-resource.yaml** file to use your custom config maps and secrets.
   Minimal **my-rhdh-custom-resource.yaml** custom resource example:

   ```
   apiVersion: rhdh.redhat.com/v1alpha3
   kind: Backstage
   metadata:
     name: my-rhdh-custom-resource
   spec:
     application:
       appConfig:
         mountPath: /opt/app-root/src
         configMaps:
            - name: my-rhdh-app-config
       extraEnvs:
         secrets:
            - name: <my_product_secrets>
       extraFiles:
         mountPath: /opt/app-root/src
       route:
         enabled: true
     database:
       enableLocalDb: true
   ```

   my-rhdh-custom-resource.yaml custom resource example with dynamic plugins and RBAC policies config maps, and external PostgreSQL database secrets:

   ```
   apiVersion: rhdh.redhat.com/v1alpha3
   kind: Backstage
   metadata:
     name: <my-rhdh-custom-resource>
   spec:
     application:
       appConfig:
         mountPath: /opt/app-root/src
         configMaps:
            - name: my-rhdh-app-config
            - name: rbac-policies
       dynamicPluginsConfigMapName: dynamic-plugins-rhdh
       extraEnvs:
         secrets:
   ```

```
        - name: <my_product_secrets>
        - name: my-rhdh-database-secrets
    extraFiles:
      mountPath: /opt/app-root/src
      secrets:
        - name: my-rhdh-database-certificates-secrets
          key: postgres-crt.pem, postgres-ca.pem, postgres-key.key
    route:
      enabled: true
  database:
    enableLocalDb: false
```

Mandatory fields

No fields are mandatory. You can create an empty Backstage CR and run Developer Hub with the default configuration.

Optional fields

spec.application.appConfig.configMaps

Enter your config map name list.

Mount files in the **my-rhdh-app-config** config map:

```
spec:
  application:
    appConfig:
      mountPath: /opt/app-root/src
      configMaps:
         - name: my-rhdh-app-config
```

Mount files in the **my-rhdh-app-config** and **rbac-policies** config maps:

```
spec:
  application:
    appConfig:
      mountPath: /opt/app-root/src
      configMaps:
         - name: my-rhdh-app-config
         - name: rbac-policies
```

spec.application.extraEnvs.envs

Optionally, enter your additional environment variables that are not secrets, such as your proxy environment variables.
Inject your **HTTP_PROXY**, **HTTPS_PROXY** and **NO_PROXY** environment variables:

```
spec:
  application:
    extraEnvs:
      envs:
        - name: HTTP_PROXY
          value: 'http://10.10.10.105:3128'
        - name: HTTPS_PROXY
          value: 'http://10.10.10.106:3128'
        - name: NO_PROXY
          value: 'localhost,example.org'
```

spec.application.extraEnvs.secrets

Enter your environment variables secret name list.
Inject the environment variables in your Red Hat Developer Hub secret:

```
spec:
  application:
    extraEnvs:
      secrets:
        - name: <my_product_secrets>
```

Inject the environment variables in the Red Hat Developer Hub and **my-rhdh-database-secrets** secrets:

```
spec:
  application:
    extraEnvs:
      secrets:
        - name: <my_product_secrets>
        - name: my-rhdh-database-secrets
```

NOTE

<my_product_secrets> is your preferred Developer Hub secret name, specifying the identifier for your secret configuration within Developer Hub.

spec.application.extraFiles.secrets

Enter your certificates files secret name and files list.
Mount the **postgres-crt.pem**, **postgres-ca.pem**, and **postgres-key.key** files contained in the **my-rhdh-database-certificates-secrets** secret:

```
spec:
  application:
    extraFiles:
      mountPath: /opt/app-root/src
      secrets:
       - name: my-rhdh-database-certificates-secrets
         key: postgres-crt.pem, postgres-ca.pem, postgres-key.key
```

spec.database.enableLocalDb

Enable or disable the local PostgreSQL database.
Disable the local PostgreSQL database generation to use an external postgreSQL database:

```
spec:
  database:
    enableLocalDb: false
```

On a development environment, use the local PostgreSQL database:

```
spec:
```

```
database:
  enableLocalDb: true
```

spec.deployment

Optionally, enter your deployment configuration.

2. Apply your Backstage CR to start or update your Developer Hub instance:

```
$ oc apply --filename=my-rhdh-custom-resource.yaml --namespace=my-rhdh-project
```

1.5. EXPOSING YOUR OPERATOR-BASED RED HAT DEVELOPER HUB INSTANCE ON MICROSOFT AZURE KUBERNETES SERVICE (AKS)

On Microsoft Azure Kubernetes Service (AKS), to expose your Red Hat Developer Hub instance, Kubernetes ingresses replace OpenShift Container Platform routes. The Red Hat Developer Hub operator does not create ingresses. Therefore, to access your Developer Hub instance via a domain name, create the required ingresses on AKS and point your domain name to it.

Prerequisites

- You have installed Red Hat Developer Hub by using the Red Hat Developer Hub Operator.

Procedure

1. Create an Ingress manifest file, named **rhdh-ingress.yaml**, specifying your Developer Hub service name as follows:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-rhdh-ingress
  namespace: my-rhdh-project
spec:
  ingressClassName: webapprouting.kubernetes.azure.com
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: my-rhdh-custom-resource
                port:
                  name: http-backend
```

2. To deploy the created Ingress, run the following command:

```
$ kubectl -n my-rhdh-project apply -f rhdh-ingress.yaml
```

Verification

1. Access the deployed Developer Hub using the URL: **https://<app_address>**, where *<app_address>* is the Ingress address obtained earlier (for example, **https://108.141.70.228**).

CHAPTER 2. DEPLOYING DEVELOPER HUB ON AKS WITH THE HELM CHART

You can deploy your Developer Hub application on Azure Kubernetes Service (AKS) to access a comprehensive solution for building, testing, and deploying applications.

Prerequisites

- You have a Microsoft Azure account with active subscription.
- You have installed the Azure CLI.
- You have installed the **kubectl** CLI.
- You are logged into your cluster using **kubectl**, and have **developer** or **admin** permissions.
- You have installed Helm 3 or the latest.
- Make sure that your system meets the minimum sizing requirements. See Sizing requirements for Red Hat Developer Hub.

Comparison of AKS specifics with the base Developer Hub deployment

- **Permissions issue**: Developer Hub containers might encounter permission-related errors, such as **Permission denied** when attempting certain operations. This error can be addressed by adjusting the **fsGroup** in the **PodSpec.securityContext**.
- **Ingress configuration**: In AKS, configuring ingress is essential for accessing the installed Developer Hub instance. Accessing the Developer Hub instance requires enabling the Routing add-on, an NGINX-based Ingress Controller, using the following command:

```
az aks approuting enable --resource-group <your_ResourceGroup> --name
<your_ClusterName>
```

TIP

You might need to install the Azure CLI extension **aks-preview**. If the extension is not installed automatically, you might need to install it manually using the following command:

```
az extension add --upgrade -n aks-preview --allow-preview true
```

NOTE

After you install the Ingress Controller, the **app-routing-system** namespace with the Ingress Controller will be deployed in your cluster. Note the address of your Developer Hub application from the installed Ingress Controller (for example, 108.141.70.228) for later access to the Developer Hub application, later referenced as **<app_address>**.

```
kubectl get svc nginx --namespace app-routing-system -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

- **Namespace management**: You can create a dedicated namespace for Developer Hub deployment in AKS using the following command:

  ```
  kubectl create namespace <your_namespace>
  ```

Procedure

1. Log in to AKS by running the following command:

   ```
   az login [--tenant=<optional_directory_name>]
   ```

2. Create a resource group by running the following command:

   ```
   az group create --name <resource_group_name> --location <location>
   ```

   TIP

   You can list available regions by running the following command:

   ```
   az account list-locations -o table
   ```

3. Create an AKS cluster by running the following command:

   ```
   az aks create \
   --resource-group <resource_group_name> \
   --name <cluster_name> \
   --enable-managed-identity \
   --generate-ssh-keys
   ```

   You can refer to **--help** for additional options.

4. Connect to your cluster by running the following command:

   ```
   az aks get-credentials --resource-group <resource_group_name> --name <cluster_name>
   ```

   The previous command configures the Kubernetes client and sets the current context in the **kubeconfig** to point to your AKS cluster.

5. Open terminal and run the following command to add the Helm chart repository:

   ```
   helm repo add openshift-helm-charts https://charts.openshift.io/
   ```

6. Create and activate the *<my-rhdh-project>* namespace:

   ```
   DEPLOYMENT_NAME=<redhat-developer-hub>
   NAMESPACE=<rhdh>
   kubectl create namespace ${NAMESPACE}
   kubectl config set-context --current --namespace=${NAMESPACE}
   ```

7. Create a pull secret, which is used to pull the Developer Hub images from the Red Hat Container Registry, by running the following command:

```
kubectl -n $NAMESPACE create secret docker-registry rhdh-pull-secret \
  --docker-server=registry.redhat.io \
  --docker-username=<redhat_user_name> \
  --docker-password=<redhat_password> \
  --docker-email=<email>
```

8. Create a file named **values.yaml** using the following template:

```
global:
  host: <app_address>
route:
  enabled: false
upstream:
  ingress:
    enabled: true
    className: webapprouting.kubernetes.azure.com
    host:
  backstage:
    image:
      pullSecrets:
        - rhdh-pull-secret
    podSecurityContext:
      fsGroup: 3000
  postgresql:
    image:
      pullSecrets:
        - rhdh-pull-secret
    primary:
      podSecurityContext:
        enabled: true
        fsGroup: 3000
  volumePermissions:
    enabled: true
```

9. To install Developer Hub by using the Helm chart, run the following command:

```
helm -n $NAMESPACE install -f values.yaml $DEPLOYMENT_NAME openshift-helm-charts/redhat-developer-hub --version 1.8.0
```

10. Verify the deployment status:

```
kubectl get deploy $DEPLOYMENT_NAME -n $NAMESPACE
```

11. Configure your Developer Hub Helm chart instance with the Developer Hub database password and router base URL values from your cluster:

```
PASSWORD=$(kubectl get secret redhat-developer-hub-postgresql -o jsonpath="{.data.password}" | base64 -d)
CLUSTER_ROUTER_BASE=$(kubectl get route console -n openshift-console -o=jsonpath='{.spec.host}' | sed 's/^[^.]*\.//')
helm upgrade $DEPLOYMENT_NAME -i "https://github.com/openshift-helm-charts/charts/releases/download/redhat-redhat-developer-hub-1.8.0/redhat-developer-hub-1.8.0.tgz" \
  --set global.clusterRouterBase="$CLUSTER_ROUTER_BASE" \
  --set global.postgresql.auth.password="$PASSWORD"
```

12. Display the running Developer Hub instance URL, by running the following command:

```
echo "https://$DEPLOYMENT_NAME-$NAMESPACE.$CLUSTER_ROUTER_BASE"
```

Verification

- Open the running Developer Hub instance URL in your browser to use Developer Hub.

Upgrade

- To upgrade the deployment, run the following command:

```
helm upgrade $DEPLOYMENT_NAME -i https://github.com/openshift-helm-
charts/charts/releases/download/redhat-redhat-developer-hub-1.8.0/redhat-developer-hub-
1.8.0.tgz
```

Delete

- To delete the deployment, run the following command:

```
helm -n $NAMESPACE delete $DEPLOYMENT_NAME
```