JupyterHub

NetApp Solutions

NetApp
August 14, 2024

Table of Contents

JupyterHub

JupyterHub Deployment

This section describes the tasks that you must complete to deploy JupyterHub in your Kubernetes cluster.

> It is possible to deploy JupyterHub on platforms other than Kubernetes. Deploying JupyterHub on platforms other than Kubernetes is outside of the scope of this solution.

Prerequisites

Before you perform the deployment exercise that is outlined in this section, we assume that you have already performed the following tasks:

1. You already have a working Kubernetes cluster.
2. You have already installed and configured NetApp Astra Trident in your Kubernetes cluster. For more details on Astra Trident, refer to the Astra Trident documentation.

Install Helm

JupyterHub is deployed using Helm, a popular package manager for Kubernetes. Before you deploy JupyterHub, you must install Helm on your Kubernetes control node. To install Helm, follow the installation instructions in the official Helm documentation.

Set Default Kubernetes StorageClass

Before you deploy JupyterHub, you must designate a default StorageClass within your Kubernetes cluster. To designate a default StorageClass within your cluster, follow the instructions outlined in the Kubeflow Deployment section. If you have already designated a default StorageClass within your cluster, then you can skip this step.

Deploy JupyterHub

After completing the steps above, you are now ready to deploy JupyterHub. JupyterHub deployment requires the following steps:

Configure JupyterHub Deployment

Before deployment it is a good practice to optimize the JupyterHub deployment for your respective environment. You can create a **config.yaml** file and utilize it during deployment using the Helm chart.

An example **config.yaml** file can be found at https://github.com/jupyterhub/zero-to-jupyterhub-k8s/blob/HEAD/jupyterhub/values.yaml

> In this config.yaml file, you can set the **(singleuser.storage.dynamic.storageClass)** parameter for the NetApp Trident StorageClass. This is the storage class that will be used to provision the volumes for individual user workspaces.

Adding Shared Volumes

If you want to use a shared volume for all JupyterHub users you can adjust your **config.yaml** accordingly. For example, if you have a shared PersistentVolumeClaim called jupyterhub-shared-volume you could mount it as /home/shared in all user pods as:

```
singleuser:
  storage:
    extraVolumes:
      - name: jupyterhub-shared
        persistentVolumeClaim:
          claimName: jupyterhub-shared-volume
    extraVolumeMounts:
      - name: jupyterhub-shared
        mountPath: /home/shared
```

This is an optional step, you can adjust these parameters to your needs.

Deploy JupyterHub with Helm Chart

Make Helm aware of the JupyterHub Helm chart repository.

```
helm repo add jupyterhub https://hub.jupyter.org/helm-chart/
helm repo update
```

This should show output like:

```
Hang tight while we grab the latest from your chart repositories...
...Skip local chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "jupyterhub" chart repository
Update Complete. □ Happy Helming!□
```

Now install the chart configured by your config.yaml by running this command from the directory that contains your config.yaml:

```
helm upgrade --cleanup-on-fail \
  --install my-jupyterhub jupyterhub/jupyterhub \
  --namespace my-namespace \
  --create-namespace \
  --values config.yaml
```

In this example:

<helm-release-name> is set to my-jupyterhub, which will be the name of your JupyterHub release.
<k8s-namespace> is set to my-namespace, which is the namespace where you want to install JupyterHub.
The --create-namespace flag is used to create the namespace if it does not already exist.
The --values flag specifies the config.yaml file that contains your desired configuration options.

Check Deployment

While step 2 is running, you can see the pods being created from the following command:

```
kubectl get pod --namespace <k8s-namespace>
```

Wait for the hub and proxy pod to enter the Running state.

```
NAME                        READY     STATUS    RESTARTS   AGE
hub-5d4ffd57cf-k68z8        1/1       Running   0          37s
proxy-7cb9bc4cc-9bdlp       1/1       Running   0          37s
```

Access JupyterHub

Find the IP we can use to access the JupyterHub. Run the following command until the EXTERNAL-IP of the proxy-public service is available like in the example output.

> We used NodePort service in our config.yaml file, you can adjust for your environment based on your setup (e.g LoadBalancer).

```
kubectl --namespace <k8s-namespace> get service proxy-public
```

```
NAME            TYPE            CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
proxy-public    NodePort    10.51.248.230    104.196.41.97    80:30000/TCP
1m
```

To use JupyterHub, enter the external IP for the proxy-public service in to a browser.

Use the NetApp DataOps Toolkit with JupyterHub

The NetApp DataOps Toolkit for Kubernetes can be used in conjunction with JupyterHub. Using the NetApp DataOps Toolkit with JupyterHub enables end users to create volume snapshots for workspace backup and/or dataset-to-model traceability directly from within a Jupyter Notebook.

Initial Setup

Before you can use the DataOps Toolkit with JupyterHub, you must grant appropriate permissions to the Kubernetes service account that JupyterHub assigns to individual user Jupyter Notebook Server pods. JupyterHub uses the service account that is specified by the `singleuser.serviceAccountName` variable in your JupyterHub Helm chart configuration file.

Create Cluster Role for DataOps Toolkit

First, create a cluster role named 'netapp-dataops' that has the required Kubernetes API permissions for creating volume snapshots.

```
$ vi clusterrole-netapp-dataops-snapshots.yaml
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: netapp-dataops-snapshots
rules:
- apiGroups: [""]
  resources: ["persistentvolumeclaims", "persistentvolumeclaims/status",
"services"]
  verbs: ["get", "list"]
- apiGroups: ["snapshot.storage.k8s.io"]
  resources: ["volumesnapshots", "volumesnapshots/status",
"volumesnapshotcontents", "volumesnapshotcontents/status"]
  verbs: ["get", "list", "create"]

$ kubectl create -f clusterrole-netapp-dataops-snapshots.yaml
clusterrole.rbac.authorization.k8s.io/netapp-dataops-snapshots created
```

Assign Cluster Role to Notebook Server Service Account

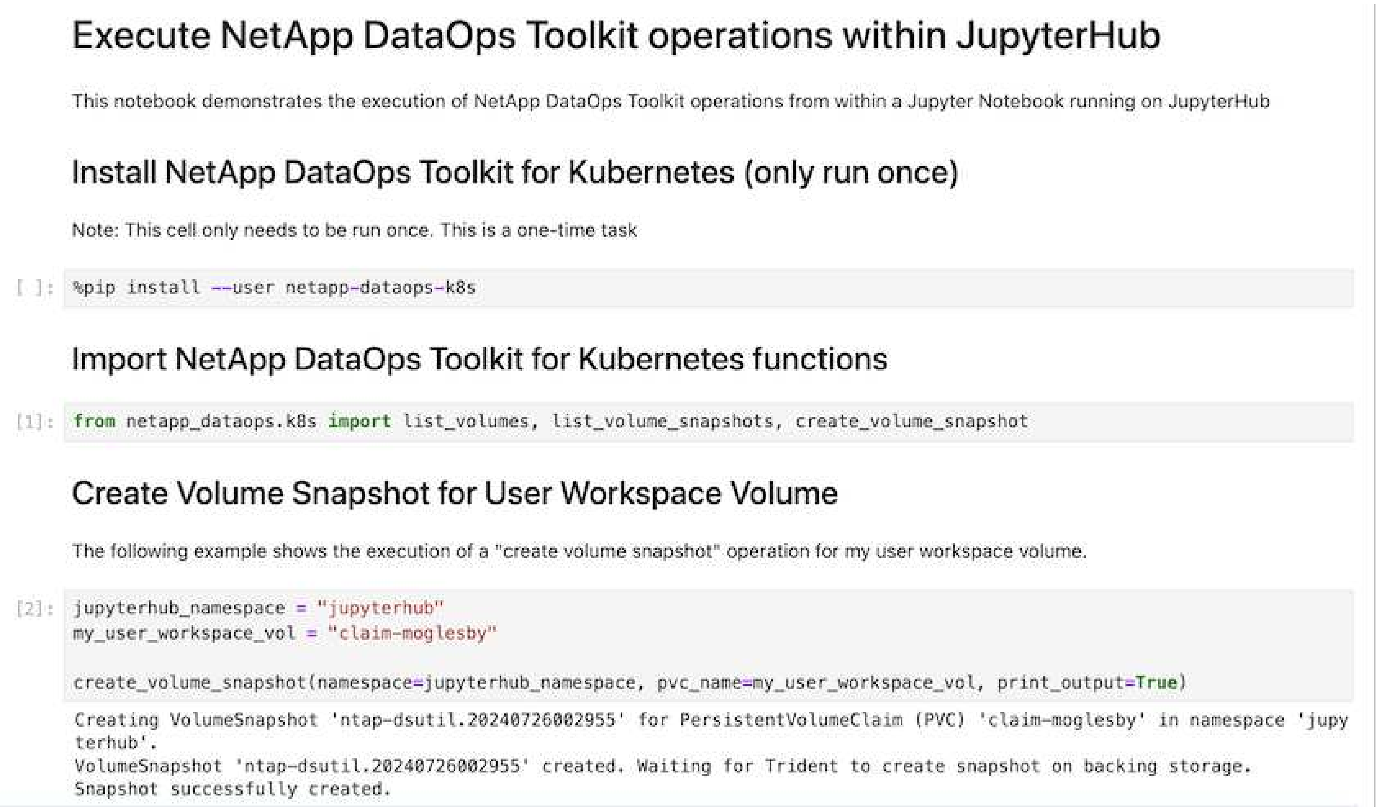
Create a role binding that assigns the 'netapp-dataops-snapshots' cluster role to the appropriate service account in the appropriate namespace. For example, if you installed JupyterHub in the 'jupyterhub' namespace, and you specified the 'default' service account via the `singleuser.serviceAccountName` variable, you would assign the the 'netapp-dataops-snapshots' cluster role to the 'default' service account in the 'jupyterhub' namespace as shown in the following example.

```
$ vi rolebinding-jupyterhub-netapp-dataops-snapshots.yaml
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jupyterhub-netapp-dataops-snapshots
  namespace: jupyterhub # Replace with you JupyterHub namespace
subjects:
- kind: ServiceAccount
  name: default # Replace with your JupyterHub
singleuser.serviceAccountName
  namespace: jupyterhub # Replace with you JupyterHub namespace
roleRef:
  kind: ClusterRole
  name: netapp-dataops-snapshots
  apiGroup: rbac.authorization.k8s.io

$ kubectl create -f ./rolebinding-jupyterhub-netapp-dataops-snapshots.yaml
rolebinding.rbac.authorization.k8s.io/jupyterhub-netapp-dataops-snapshots
created
```

Create Volume Snapshots Within Jupyter Notebook

Now, JupyterHub users can use the NetApp DataOps Toolkit to create volume snapshots directly from within a Jupyter Notebook as shown in the following example.



Execute NetApp DataOps Toolkit operations within JupyterHub

This notebook demonstrates the execution of NetApp DataOps Toolkit operations from within a Jupyter Notebook running on JupyterHub

Install NetApp DataOps Toolkit for Kubernetes (only run once)

Note: This cell only needs to be run once. This is a one-time task

```
[ ]: %pip install --user netapp-dataops-k8s
```

Import NetApp DataOps Toolkit for Kubernetes functions

```
[1]: from netapp_dataops.k8s import list_volumes, list_volume_snapshots, create_volume_snapshot
```

Create Volume Snapshot for User Workspace Volume

The following example shows the execution of a "create volume snapshot" operation for my user workspace volume.

```
[2]: jupyterhub_namespace = "jupyterhub"
     my_user_workspace_vol = "claim-moglesby"

     create_volume_snapshot(namespace=jupyterhub_namespace, pvc_name=my_user_workspace_vol, print_output=True)

     Creating VolumeSnapshot 'ntap-dsutil.20240726002955' for PersistentVolumeClaim (PVC) 'claim-moglesby' in namespace 'jupy
     terhub'.
     VolumeSnapshot 'ntap-dsutil.20240726002955' created. Waiting for Trident to create snapshot on backing storage.
     Snapshot successfully created.
```

Copyright information

Trademark information