Rapidly Clone a Dataset to create a Data Scientist Workspace

NetApp Solutions

Dorian Henderson, Kevin Hoke
January 21, 2021

Table of Contents

Rapidly Clone a Dataset to create a Data Scientist Workspace

The example DAG outlined in this section implements a workflow that takes advantage of NetApp FlexClone technology to clone a dataset volume rapidly and efficiently and create a data scientist or developer workspace.

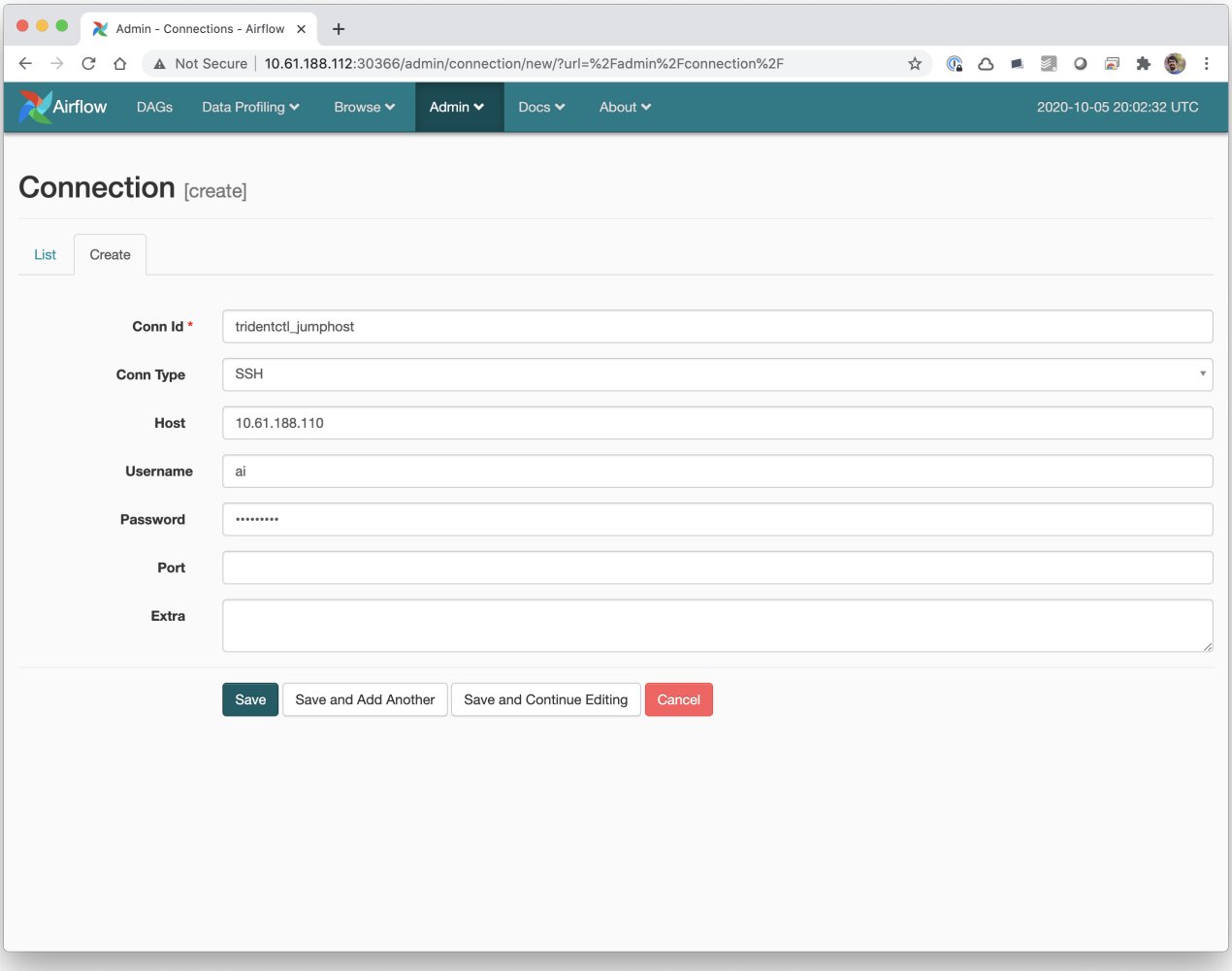
Prerequisites

For this DAG to function correctly, you must complete the following prerequisites:

1. You must have created a connection in Airflow for your ONTAP system as outlined in Prerequisite #1 in the section “Implement an End-to-End AI Training Workflow with Built-in Traceability and Versioning.”
2. You must have created a connection in Airflow for a host that is accessible via SSH and on which `tridentctl`, the NetApp Trident management utility, is installed and configured to point to your Kubernetes cluster.

   To manage connections in Airflow, navigate to Admin > Connections in the Airflow web service UI. The example screenshot that follows shows the creation of a connection for a specific host on which `tridentctl` is installed and configured. The following values are required:

   - **Conn ID.** Unique name for the connection.
   - **Conn Type.** Must be set to ‘SSH’.
   - **Host.** The host name or IP address of the host.
   - **Login.** Username to use when accessing the host via SSH.
   - **Password.** Password to use when accessing the host via SSH.



1. There must be an existing PersistentVolumeClaim (PVC) within your Kubernetes cluster that is tied to the volume that contains the dataset that you wish to clone.

DAG Definition

The Python code excerpt that follows contains the definition for the example DAG. Before executing this example DAG in your environment, you must modify the parameter values in the `DEFINE PARAMETERS` section to match your environment.

```
# Airflow DAG Definition: Create Data Scientist Workspace
#
# Steps:
#   1. Clone source volume
#   2. Import clone into Kubernetes using Trident
from airflow.utils.dates import days_ago
from airflow.secrets import get_connections
from airflow.models import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.contrib.operators.ssh_operator import SSHOperator
```

```
from datetime import datetime
##### DEFINE PARAMETERS: Modify parameter values in this section to match
your environment #####
## Define default args for DAG
create_data_scientist_workspace_dag_default_args = {
    'owner': 'NetApp'
}
## Define DAG details
create_data_scientist_workspace_dag = DAG(
    dag_id='create_data_scientist_workspace',
    default_args=create_data_scientist_workspace_dag_default_args,
    schedule_interval=None,
    start_date=days_ago(2),
    tags=['dev-workspace']
)
## Define volume details (change values as necessary to match your
environment)
# ONTAP system details
ontapAirflowConnectionName = 'ontap_ai'  # Name of the Airflow connection
that contains connection details for your ONTAP system's cluster admin
account
verifySSLCert = False   # Denotes whether or not to verify the SSL cert
when calling the ONTAP API
# Source volume details
sourcePvName = 'pvc-79e0855a-30a1-4f63-b34c-1029b1df49f6' # Name of
Kubernetes PV corresponding to source volume
# Clone volume details (details for the new clone that you will be
creating)
timestampForVolumeName = datetime.today().strftime("%Y%m%d_%H%M%S")
cloneVolumeName = 'airflow_clone_%s' % timestampForVolumeName
clonePvcNamespace = 'airflow'   # Kubernetes namespace that you want the
new clone volume to be imported into
## Define tridentctl jumphost details (change values as necessary to match
your environment)
tridentctlAirflowConnectionName = 'tridentctl_jumphost' # Name of the
Airflow connection of type 'ssh' that contains connection details for a
jumphost on which tridentctl is installed
## Define Trident details
tridentStorageClass = 'ontap-flexvol'   # Kubernetes StorageClass that you
want to use when importing the new clone volume
tridentNamespace = 'trident'    # Namespace that Trident is installed in
tridentBackend = 'ontap-flexvol'    # Trident backend that you want to use
when importing the new clone volume
#########################################################################
######################
# Define function that clones a NetApp volume
```

```
def netappClone(task_instance, **kwargs) -> str :
    # Parse args
    for key, value in kwargs.items() :
        if key == 'sourcePvName' :
            sourcePvName = value
        elif key == 'verifySSLCert' :
            verifySSLCert = value
        elif key == 'airflowConnectionName' :
            airflowConnectionName = value
        elif key == 'cloneVolumeName' :
            cloneVolumeName = value
    # Install netapp_ontap package
    import sys, subprocess
    result = subprocess.check_output([sys.executable, '-m', 'pip',
'install', '--user', 'netapp-ontap'])
    print(str(result).replace('\\n', '\n'))

    # Import needed functions/classes
    from netapp_ontap import config as netappConfig
    from netapp_ontap.host_connection import HostConnection as
NetAppHostConnection
    from netapp_ontap.resources import Volume, Snapshot
    from datetime import datetime
    import json
    # Retrieve ONTAP cluster admin account details from Airflow connection
    connections = get_connections(conn_id = airflowConnectionName)
    ontapConnection = connections[0]    # Assumes that you only have one
connection with the specified conn_id configured in Airflow
    ontapClusterAdminUsername = ontapConnection.login
    ontapClusterAdminPassword = ontapConnection.password
    ontapClusterMgmtHostname = ontapConnection.host

    # Configure connection to ONTAP cluster/instance
    netappConfig.CONNECTION = NetAppHostConnection(
        host = ontapClusterMgmtHostname,
        username = ontapClusterAdminUsername,
        password = ontapClusterAdminPassword,
        verify = verifySSLCert
    )

    # Convert pv name to ONTAP volume name
    # The following will not work if you specified a custom storagePrefix
when creating your
    #   Trident backend. If you specified a custom storagePrefix, you will
need to update this
    #   code to match your prefix.
```

```
    sourceVolumeName = 'trident_%s' % sourcePvName.replace("-", "_")
    print('\nSource pv name: ', sourcePvName)
    print('Source ONTAP volume name: ', sourceVolumeName)
    # Create clone
    sourceVolume = Volume.find(name = sourceVolumeName)
    cloneVolume = Volume.from_dict({
        'name': cloneVolumeName,
        'svm': sourceVolume.to_dict()['svm'],
        'clone': {
            'is_flexclone':'true',
            'parent_volume': sourceVolume.to_dict()
        },
        'nas': {
            'path': '/%s' % cloneVolumeName
        }
    })
    response = cloneVolume.post()
    print("\nAPI Response:")
    print(response.http_response.text)
    # Retrieve clone volume details
    cloneVolume.get()
    # Convert clone volume details to JSON string
    cloneVolumeDetails = cloneVolume.to_dict()
    print("\nClone Volume Details:")
    print(json.dumps(cloneVolumeDetails, indent=2))
    # Create PVC name that resembles volume name and push as XCom for
future use
    task_instance.xcom_push(key = 'clone_pvc_name', value =
cloneVolumeDetails['name'].replace('_', '-'))
    # Return name of new clone volume
    return cloneVolumeDetails['name']
# Define DAG steps/workflow
with create_data_scientist_workspace_dag as dag :
    # Define step to clone source volume
    clone_source = PythonOperator(
        task_id='clone-source',
        provide_context=True,
        python_callable=netappClone,
        op_kwargs={
            'airflowConnectionName': ontapAirflowConnectionName,
            'sourcePvName': sourcePvName,
            'verifySSLCert': verifySSLCert,
            'cloneVolumeName': cloneVolumeName
        },
        dag=dag
    )
```

```
    # Define step to import clone into Kubernetes using Trident
    cloneVolumeName = "{{ task_instance.xcom_pull(task_ids='clone-source',
key='return_value') }}"
    clonePvcName = "{{ task_instance.xcom_pull(task_ids='clone-source',
key='clone_pvc_name') }}"
    import_command = '''cat << EOD > import-pvc-%s.yaml && tridentctl -n
%s import volume %s %s -f ./import-pvc-%s.yaml && rm -f import-pvc-%s.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: %s
  namespace: %s
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: %s
EOD''' % (clonePvcName, tridentNamespace, tridentBackend, cloneVolumeName,
clonePvcName, clonePvcName, clonePvcName, clonePvcNamespace,
tridentStorageClass)
    import_clone = SSHOperator(
        task_id="import-clone",
        command=import_command,
        ssh_conn_id=tridentctlAirflowConnectionName
    )
    # State that the import step should be executed after the initial
clone step completes
    clone_source >> import_clone
```

Next: Trigger a SnapMirror Volume Replication Update