EVIDEN

Software Suites

jarvice-kubeflow

Administration guide

Release 1.3.0

May 12, 2025

Chapter 0. Contents:

1 Introduction **1**

2 Kubeflow Application prerequisites **3**
2.1 How to Opt-Out 3

3 Vault creation **5**

4 Launching the Kubeflow Application **7**

5 User management with Keycloak **9**
5.1 Mandatory Configurations 9
5.2 Give users access 10

6 System management with the GoTTY shell **13**

7 Accessing data, within Kubeflow, from an NFS server **15**
7.1 Mounting a PersistentVolumeClaim (PVC) 15
7.2 Important parameters to customize 18
7.3 Making a single NFS server accessible by several users 18

8 Configuring and using KServe **19**
8.1 Introduction 19
8.2 Configuring authorized paths 19
8.3 Using KServe 19

9 Pause/Resume **23**
9.1 Pausing a Jarvice Kubeflow Application 23
9.2 Resuming a Jarvice Kubeflow Application 24

10 Known bugs **25**
10.1 RBAC Access Denied Issue to access Kubeflow 25

Chapter 1. Introduction

This guide is targeted at Machine Learning Team Administrators (ML Team Admin). Its purpose is to help them configure and customize the Kubeflow application within Jarvice.

Chapter 2. Kubeflow Application prerequisites

To avoid an issue[1] in the Kubeflow deployment in which some pods enter in a crashloop state reporting `too many files open` error, it is needed to modify some sysctl values: `fs.inotify.max_user_instances` and `fs.inotify.max_user_watches` from their defaults to `1280` and `655360` respectively. This can be done directly modifying the `/etc/sysctl.conf` file in each of the hosts machines. Alternatively, there is a `sysctl-config` app that runs a DaemonSet and can modify those values automatically, so no further action from the system administrator shall be needed.

Warning:

As the containers need to modify the sysctl values, they are executed with the privileged flag set to true.

2.1 How to Opt-Out

Check with the system administrator if those changes were already made. If so, they should as well modify the Appdef to remove the reference to the sysctl-config app as described below:

```
[
  ...
  {
    "name": "nswatcher",
    "type": "kustomize",
    "path": "manifests/components/apps/nswatcher"
  },
  {
    "name": "sysctl-config",
    "type": "kustomize",
    "path": "manifests/components/apps/sysctl-config"
  }
],
```

to this:

```
[
  ...
  {
    "name": "nswatcher",
```

(continues on next page)

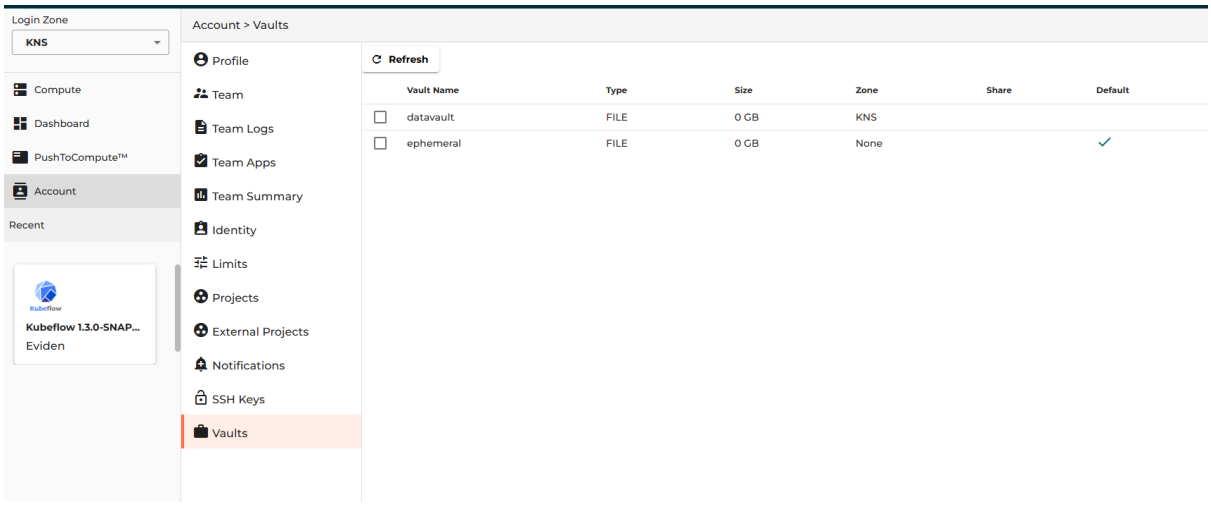
[1] https://github.com/kubeflow/manifests/issues/2087

(continued from previous page)

```
    "type": "kustomize",
    "path": "manifests/components/apps/nswatcher"
  }
],
```

Chapter 3. Vault creation

It is possible for end users of Jarvice Kubeflow to get access to the data that are stored on an NFS server (like a Filestore instance on GCP, for example). This will be done using Jarvice Vault. The Vault storage must have been created before launching the Kubeflow Application by the ML Team Admin; it is available then, in the list as shown below:



Note:

The Jarvice Admin Role is required to create Vault storage.

Note:

The csi-driver-nfs should be installed on the host to use the vault storage. see https://github.com/kubernetes-csi/csi-driver-nfs

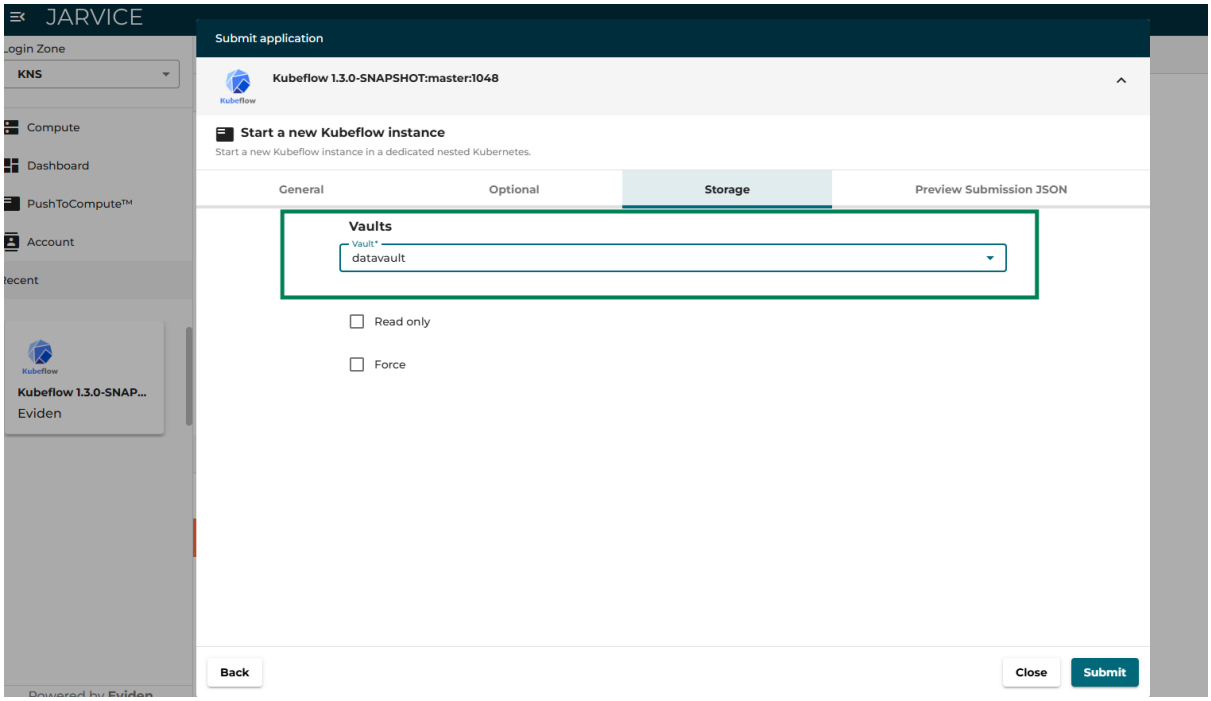
Chapter 4. Launching the Kubeflow Application

The Kubeflow application is available in Jarvice in the Menu *Compute -> All Applications*

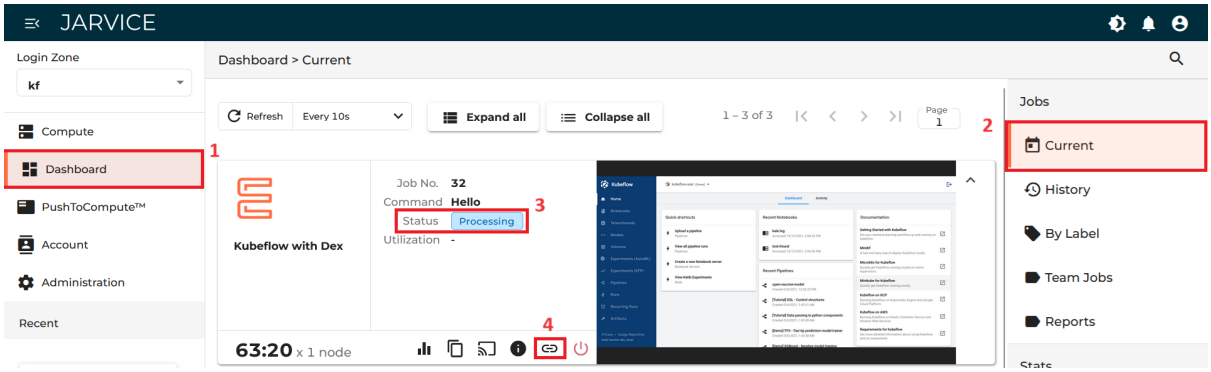
- Find the Kubeflow application
- Click the Kubeflow application tile
- Select *Start a new Kubeflow instance*
- Then, you can customize your application settings: machine type, allocated resources, admin credentials, vault storage etc…(If you want to specify the size of the dynamic storage instead of using the default value, be careful to specify the units)
- Finally, click *Submit*

Note:

Vault storage must be created in Jarvice before launching Kubeflow Application.



It takes a couple of minutes to deploy the application. You can check the deployment status in the dashboard.



Under *Dashboard (1)*, *Current (2)*, you can see the *job status (3)*. When the status is *Processing*, the application is deployed and ready. Clicking the *link button (4)* will copy to the clipboard the public URL where the Kubeflow application can be accessed. All you have to do then is to paste it in the navigation bar in the browser of your choice.

Chapter 5. User management with Keycloak

To allow end users (e.g. data scientist) to access the Kubeflow application, the ML Team Admin needs to create a Kubeflow user for each of them. This distribution of Kubeflow supports Keycloak to authenticate users.

Note:

Please note that these users are not related to Jarvice users. They are defined within Keycloak and are managed by the ML Team Admin.

Please ask the Jarvice sys admin for the Keycloak url and verify you can log in.

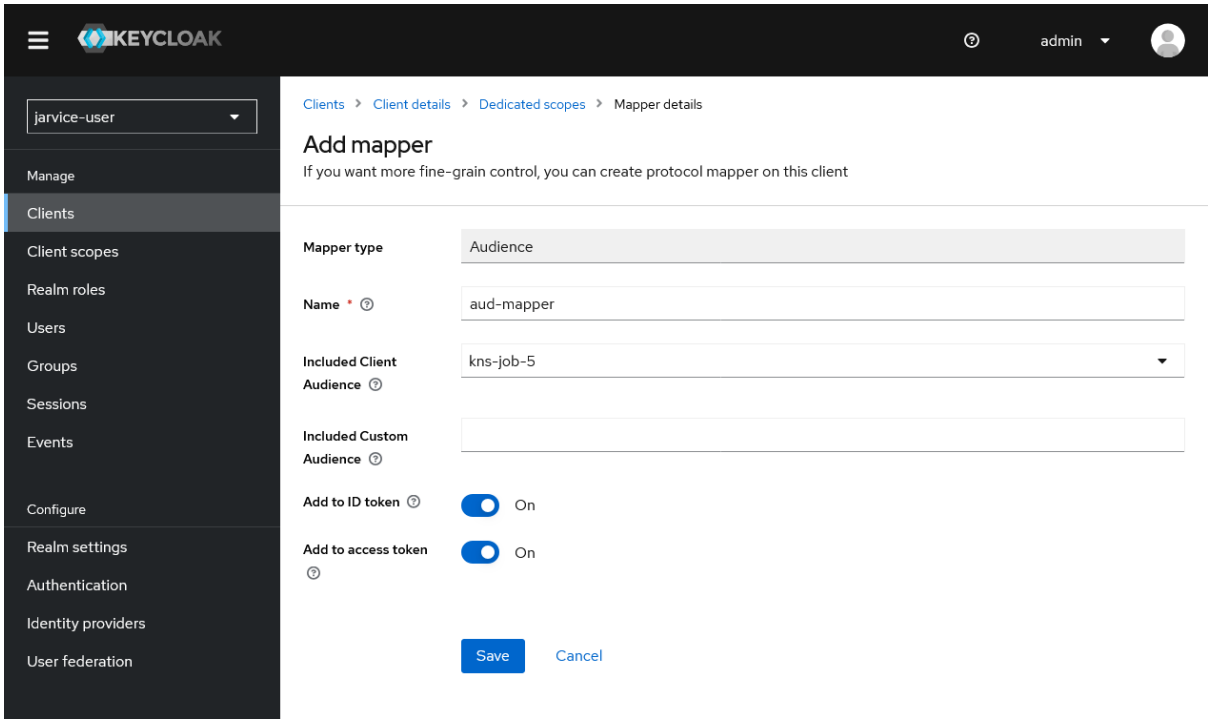
For every Kubeflow instance with Keycloak created, a `client` is added automatically to the ML Team Admin's `realm`, where the client id is kns-job-<job number> and the realm name is the username of the ML Team admin.

5.1 Mandatory Configurations

5.1.1 Audience mapper configuration

Following a Kubeflow instance creation with Keycloak, an audience mapper must be configured for Keycloak to work with the instance.

- Go to `Clients` -> `kns-job-<job number>` -> `Client scopes` -> `kns-job-<job number>-dedicated`
- Click `Configure a new mapper`
- Select `Audience`
- Fill the form:
  - Give the mapper a `Name`
  - Select the job's client in the `Included Client Audience` drop down
  - Make sure `Add to ID token` and `Add to access token` are set to `On`



5.1.2 Access token lifespan configuration

The Keycloak "Access Token Lifespan" must be set to a value greater than the "cookie_expire" duration of Oauth2-proxy. By default, access tokens in Keycloak expire after 5 minutes. Since the "cookie_expire" is configured to 2 hours, the "Access Token Lifespan" in the Keycloak client settings must be adjusted accordingly to exceed 2 hours—for instance, setting it to 130 minutes.

5.2 Give users access

Note:

To be able to give the users access, you must have the keycloak roles to manage the different settings of the realm created for the ML Team Admin (*realm-management* role).

A user will have access to Kubeflow once:

1. They have been created in the Keycloak realm, if they don't already exist
2. They are assigned the `kns-role` role of the job's client
3. Their email address is verified

Note:

You must select `Filter by clients` to see the client roles.

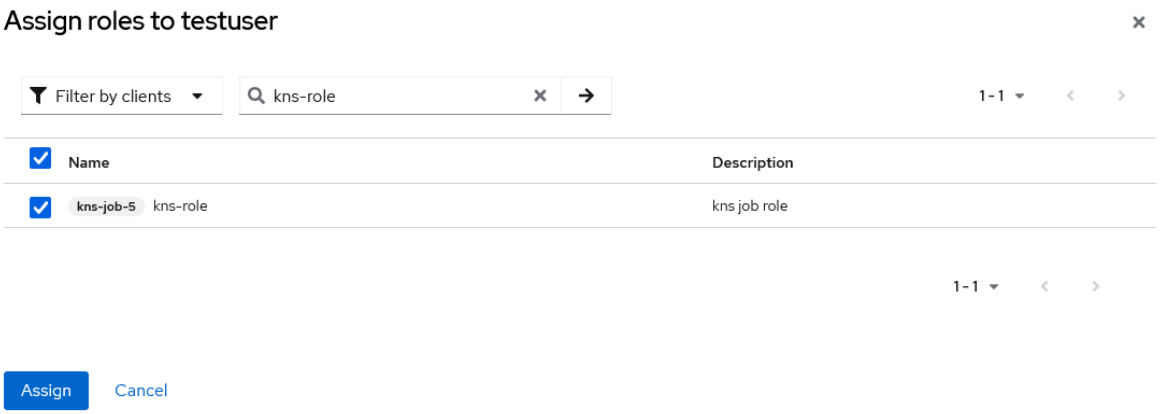


Fig. 1: Example: We assign the kns-role from the kns-job-5 client to testuser, giving them access to the Kubeflow running in job #5.

Chapter 6. System management with the GoTTY shell

The Kubeflow application is launched in a virtual cluster. The ML Team Admin can access the CLI of this cluster via an application: the GoTTY shell. This gotty shell is an optional component. It can be enabled by adding *"enable_gotty_shell": "true"* in the *"nestedkubernetes"* object of the AppDef. Please check with your Jarvice System Administrator to verify whether the shell is enabled in your Kubeflow application.

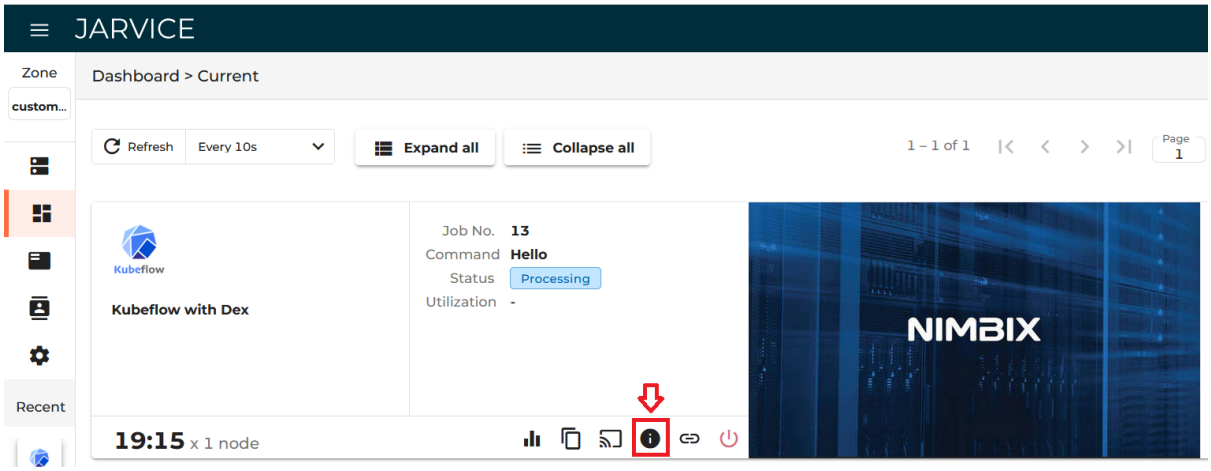
```
"nestedkubernetes": {
    "kubernetes_version": "v1.27",
    "enable_gotty_shell": "true",
    ...
```

If it is enabled, the shell is accessible at the following address:

```
https://<job-name>.<cluster-domain>/gotty-shell
```

where *<job-name>* is the name of the Jarvice job that is running the Kubeflow application.

You can retrieve the credentials for the GoTTY webshell in the info page of the application.



In the GoTTY shell, it is possible to perform basic system commands as well as *kubectl* commands in the virtual cluster. This allows the ML Team Admin to perform various tasks like debugging or managing Kubernetes resources.

Note:

It is recommended to use the Chrome browser to access the GoTTY shell.

Chapter 7. Accessing data, within Kubeflow, from an NFS server

It is possible for end users of Jarvice Kubeflow to get access to data stored on an NFS server (like a Filestore instance on GCP, for example). Vault storage must have been created before launching the Kubeflow Application by the ML Team Admin. You can check if it is available and look up the available storage classes by running the following command in the GoTTY shell (see *System management with the GoTTY shell*)

```
kubectl get storageclass
```

and you should have, at least the two following ones:

```
NAME                            PROVISIONER             RECLAIMPOLICY  ␣
↪VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
<my vault_name>                 nfs.csi.k8s.io          Retain          Immediate ␣
↪               false                   27h
<default-storage-class> (default)  rancher.io/local-path   Delete         ␣
↪WaitForFirstConsumer    false                  6d4h
```

7.1 Mounting a PersistentVolumeClaim (PVC)

To make the data stored in the Vault, accessible in Kubeflow, you need to claim it with a PVC. A PVC is associated to a single end-user. As such, it needs to be created in the user namespace.

A PVC could be created:

- by the ML Team Admin, using the manifest below in the GoTTY shell,
- by the End user: from the *Volumes* tab, in the Kubeflow dashboard or from a terminal using the manifest below from a Jupyter Notebook.

7.1.1 Creating a PVC using a manifest

Here is a manifest for creating a PVC, to be customized with your own values (see *Important parameters to customize*); it could be launched by the ML Team Admin in the GoTTY shell or the End User from a terminal of a Jupyter Notebook:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: <pvc-name>
    namespace: <user-namespace>
spec:
accessModes:
```

(continues on next page)

(continued from previous page)

```
    - ReadWriteMany
storageClassName: <vault-name>
resources:
    requests:
        storage: <pvc-capacity>
```

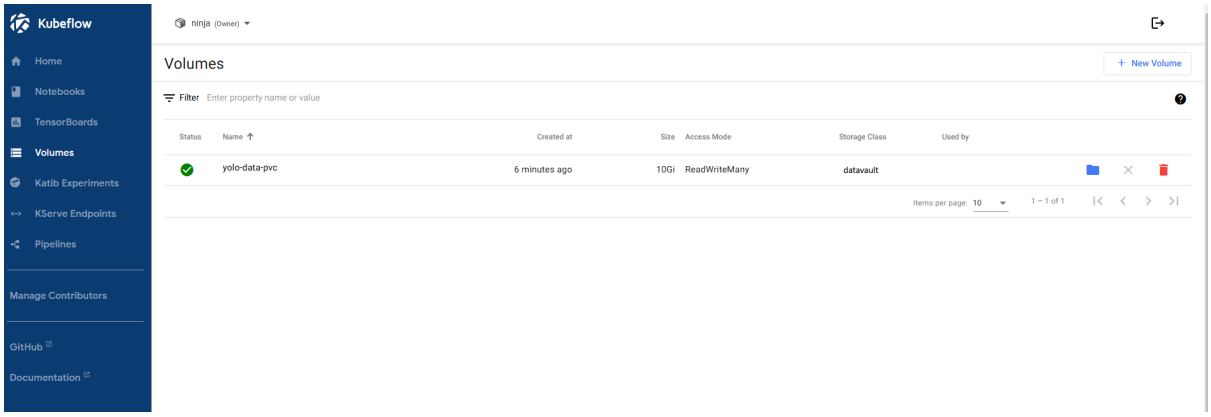
```
kubectl apply -f <pvc-manifest-file>
```

You can check that the PVC has been successfully created with

```
kubectl get pvc <pvc-name> -n <user-namespace>
```

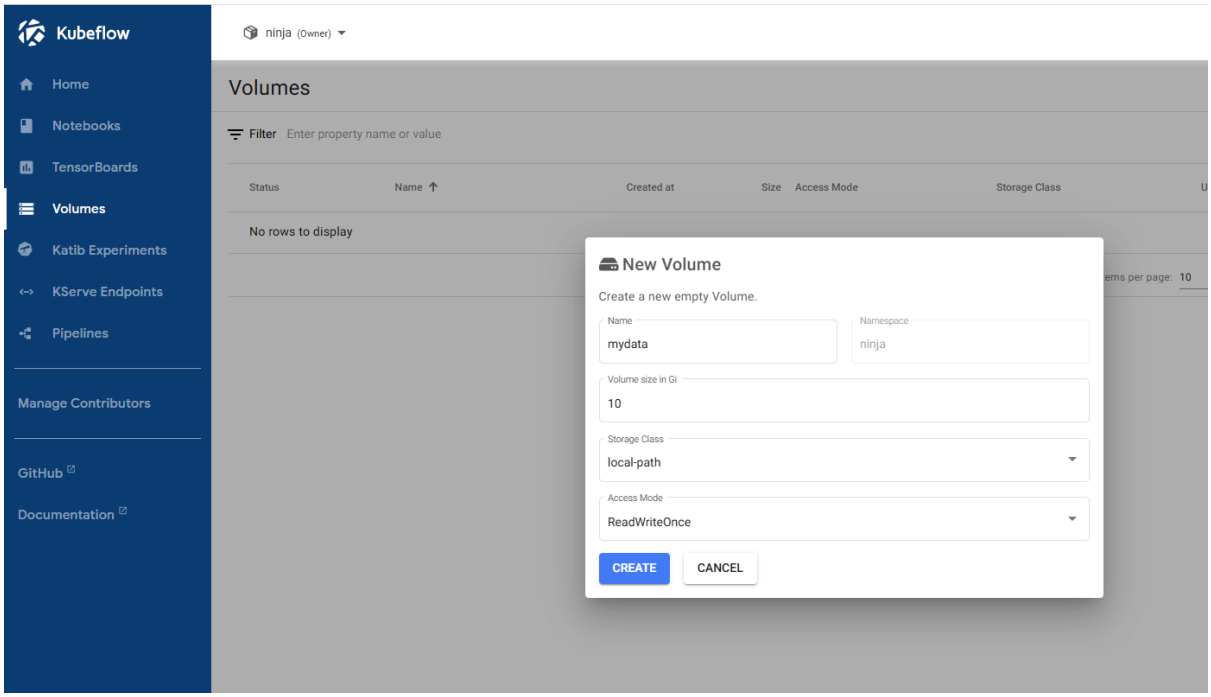
If the PVC has been correctly mounted, its status should be “Bound” and it should be visible in the Kubeflow GUI in the *Volumes* tab of its end-user. Please note that only one PVC can be bound to a PV.

```
[root@                    ~]# kubectl get pvc yolo-data-pvc -n kubeflow-user-example-com
NAME            STATUS   VOLUME        CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
yolo-data-pvc   Bound    yolo-data-pv   100Gi      RWX                           <unset>                 7d23h
```

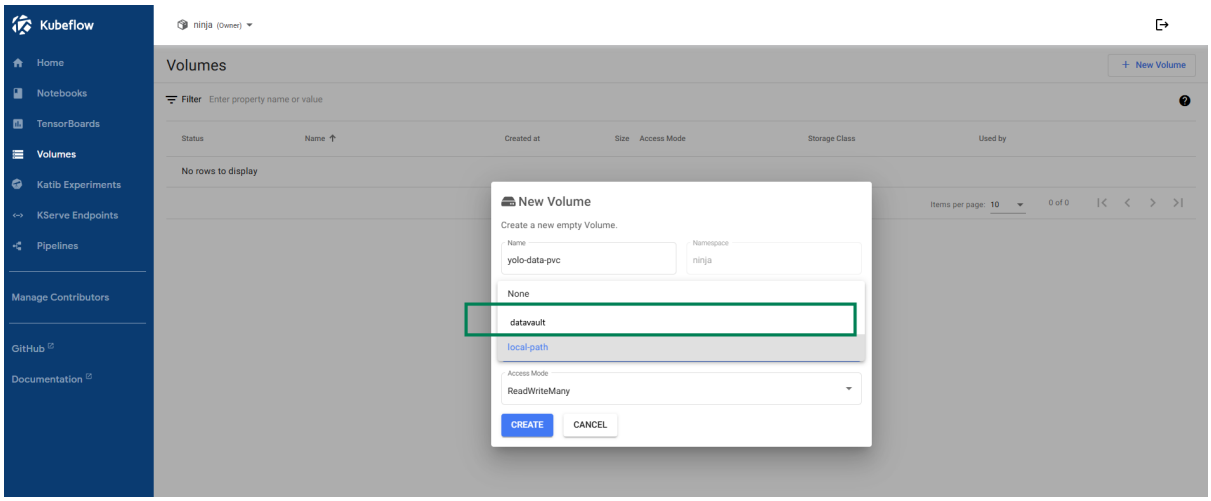


7.1.2 PVC creation with the Kubeflow Dashboard

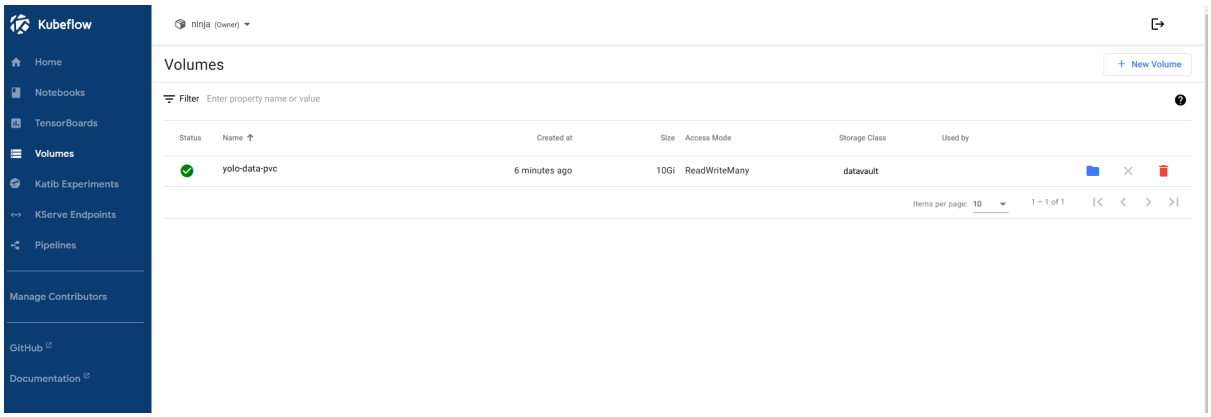
A PVC could be created from the tab *Volumes* of the Kubeflow dashboard, by selecting *New Volume*:



and then select the Vault Storage as Storage class:



it will be then available in the tab *Volumes* of the Kubeflow dashboard:



7.2 Important parameters to customize

There are two important parameters that need to be configured in order to correctly setup your PVs and PVCs, the *StorageClassName* and the *AccessModes*.

7.2.1 Storage class

You need to update the *storageClassName* in both files according to your cluster configuration.

- On clusters created on GKE with Jarvice Terraform scripts
  - By default, the storage class is “standard-rwo”. In that case, the accessModes are restricted to *ReadWriteOnce*.
  - To use other access modes, you can enable the Filestore Container Storage Interface (CSI) driver. This provides the “standard-rwx” storage class, which supports all available access modes.
- On other types of clusters, you can check the available storage classes with *kubectl get sc* and choose one that supports NFS access.

7.2.2 Access modes

AccessModes is a list of one or more modes that determine how the PVs and PVCs can be accessed. Here is a brief description of the main available modes:

- *ReadOnlyMany*: the volume can be mounted in read-only mode by several nodes
- *ReadWriteMany*: the volume can be mounted in read-write mode by several nodes
- *ReadWriteOnce*: the volume can be mounted in read-write mode by only one node (but can be accessed by several pods)
- *ReadWriteOncePod* (new): the volume can be mounted in read-write mode by only one pod. This mode is only available under certain requirements.

Please check Kubernetes documentation[2] for more information.

7.3 Making a single NFS server accessible by several users

By default, all the users of Kubeflow Application have access to an NFS server, they need to create a PVC to access it. For the PVCs, proceed as above with one PVC per storage for the desired user.

Please be aware that it is recommended to allow NFS multi-user access only on volumes with ReadOnlyMany access mode. Indeed, having several users with write access to the same volume could lead to undesired/unexpected side effects.

[2] https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Chapter 8. Configuring and using KServe

8.1 Introduction

KServe is installed as part of the Kubeflow application. To minimize the work of the ML Team Admin, the following configurations are performed automatically:

- at the application launch, *https* is set as the default protocol and the job URL is set as the external domain (in the *config-domain* ConfigMap of the *knative-serving* namespace)
- at the creation of each user namespace, a wildcard host is added to the Istio *job-ingress*, to provide external access to all the KServe endpoints created by this user

These configurations allow end-users to use KServe as soon as the Kubeflow application is deployed.

8.2 Configuring authorized paths

KServe uses Istio ingresses to provide external access to its API. All the authorized paths are declared in the *whitelist-by-paths* AuthorizationPolicy in the *istio-system* namespace. You can manage these paths by editing this policy.

```
kubectl edit authorizationpolicy/whitelist-by-paths -n istio-system
```

8.3 Using KServe

In the Kubeflow application, using KServe endpoints requires authentication. Since version 1.9, Kubeflow has replaced the OIDC authservice by the OAuth2-proxy (see Kubeflow documentation[3] for more information). Here, we provide some code samples that can be used to access KServe endpoints.

[3] https://github.com/kubeflow/manifests/tree/master/common/oauth2-proxy

8.3.1 From *curl*

In the browser, when logged in into Kubeflow, one can retrieve the *oauth2_proxy_kubeflow* cookie by opening the Developper tools (with F12 in most browsers). Then, the cookie can be passed in the requests for authentication.

For example, to get information about a model:

```
curl -k -X GET -H "Cookie: oauth2_proxy_kubeflow=<cookie-value>" -H "Content-Type:␣
↪application/json" "https://<kserve-endpoint>/v1/models/<model-name>"
```

8.3.2 From Python code

In the following example, we use simple HTTP requests to retrieve the cookie. Then, we use it to get the model information from the inference service.

```
import requests

user_login = ...
password = ...
inference_service_name = ...
user_namespace = ...
job_domain = ...
OAUTH2_COOKIE_LABEL = "oauth2_proxy_kubeflow"

def get_auth(url: str, username: str, password: str) -> dict:
    """
    Try to obtain a session cookie.

    :param url: Kubeflow OAuth2 URL, including protocol
    :param username: username
    :param password: password
    :return: session cookie in a cookie jar
    """

    # use a persistent session (for cookies)
    with requests.Session() as s:
        ################
        # Determine if Endpoint is Secured
        ################
        resp = s.get(url, allow_redirects=True, verify=False)
        if resp.status_code != 200:
            raise RuntimeError(
                f"HTTP status code '{resp.status_code}' for GET against: {url}"
            )

        redirect_url = resp.url

        ################
        # Attempt Login
        ################
        resp = s.post(
            redirect_url,
            data={"login": username, "password": password},
            allow_redirects=True,
            verify=False
        )
```

(continues on next page)

(continued from previous page)

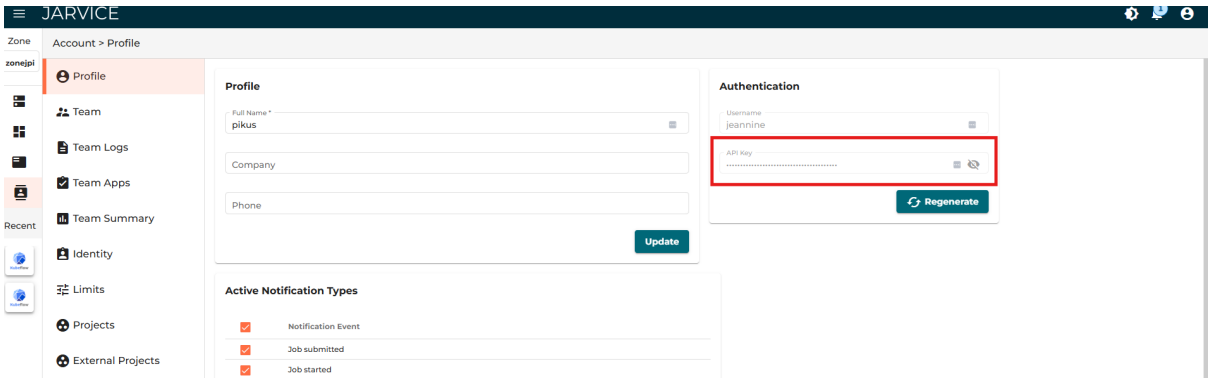
```
        if len(resp.history) == 0:
            raise RuntimeError(
                f"Login credentials were probably invalid - "
                f"No redirect after POST to: {redirect_url}"
            )

        # store the session cookies in a "key1=value1; key2=value2" string
        cookies = {OAUTH2_COOKIE_LABEL: s.cookies.get(OAUTH2_COOKIE_LABEL)}
        jar = requests.cookies.cookiejar_from_dict(cookies)

    return jar

kube_url = f"https://{job_domain}/oauth2/start"
jar = get_auth(kube_url, user_login, password)
url = f"https://{inference_service_name}.{user_namespace}.{job_domain}"
# Get and print model information
response = requests.get(url=f"{url}/v1/models/{model_name}", cookies=jar,␣
↪verify=False)
print(response.text)
```

Chapter 9. Pause/Resume

A Jarvice Kubeflow Application may not be in use at all times, and to save resources in this period of inactivity you can pause it, and later on you can resume it to start using it again. This functionality will help to save resources and so it won't be billed during the time it is paused. This functionality is not yet implemented in the Jarvice frontend, so we described a temporary solution below.

9.1 Pausing a Jarvive Kubeflow Application

To pause the kns application, we send a signal 19 (SIGSTP) to the kns job using jarvice rest API (see /jarvice/signal endpoint in https://jarvice.readthedocs.io/en/latest/api/#interact-with-job) The user is the ML Team Admn username that launched the job and the api key is available in Jarvice:



The command is the following one:

```
curl -k -X GET 'https://api.jxe.<master_node_ip>.nip.io/jarvice/signal?number=1&
↪signal=19&username=<ML Team Admin username>&apikey=<root api key>'
```

You can check:

- there is no remaining pods under the job namespace,
- you can no longer access the kubeflow instance,
- the status of the job in the jarvice dashboard is "Processing" with sub status "Suspended by user" (note that this status could change when the nimbix team will implement the frontend part).

9.2 Resuming a Jarvive Kubeflow Application

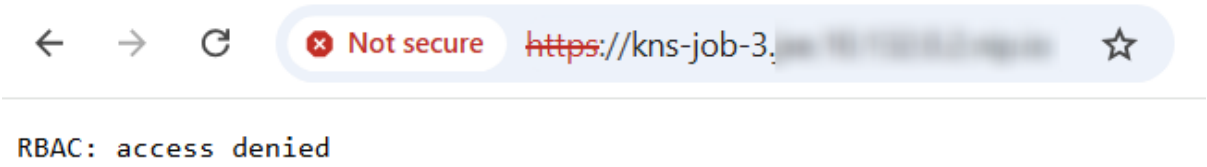
To resume the application: send a signal 18 (SIGSTP) to the kns job using jarvice rest API (see /jarvice/signal endpoint in https://jarvice.readthedocs.io/en/latest/api/#interact-with-job)

Chapter 10. Known bugs

10.1 RBAC Access Denied Issue to access Kubeflow

10.1.1 Issue Description

An “RBAC access denied” error can be encountered when attempting to access Kubeflow if you are using Gotty-Shell, as shown in the next figure



This issue occurs when there is an active Gotty-Shell session running, because the Authorization header from the Gotty-Shell session is sent to Kubeflow. Since both services are exposed through the same URL, the conflicting Authorization headers prevent proper authentication for Kubeflow access.

This behavior effectively blocks access to Kubeflow while the Gotty-Shell session remains active.

10.1.2 Workaround

To resolve the issue, the active Gotty-Shell session needs to be terminated. There are two approaches to achieve this:

1. **Delete Browser History**: Clear your browser history and cookies. This will remove the session cookies associated with the Gotty-Shell and allow you to access Kubeflow.
2. **Terminate the Gotty-Shell Session by Logging in with Incorrect Credentials**:
   - Access the Gotty-Shell URL in your browser using the following format: `<user>@<gotty-shell-url>`
   - You will be prompted for credentials. If you enter incorrect credentials, this will terminate the active session.

After following either of these steps, you should be able to access Kubeflow without encountering the “RBAC Access Denied” error.