Prepared by Anray liu, justin zhang June 30, 2025

volume cleaner

Technical Report

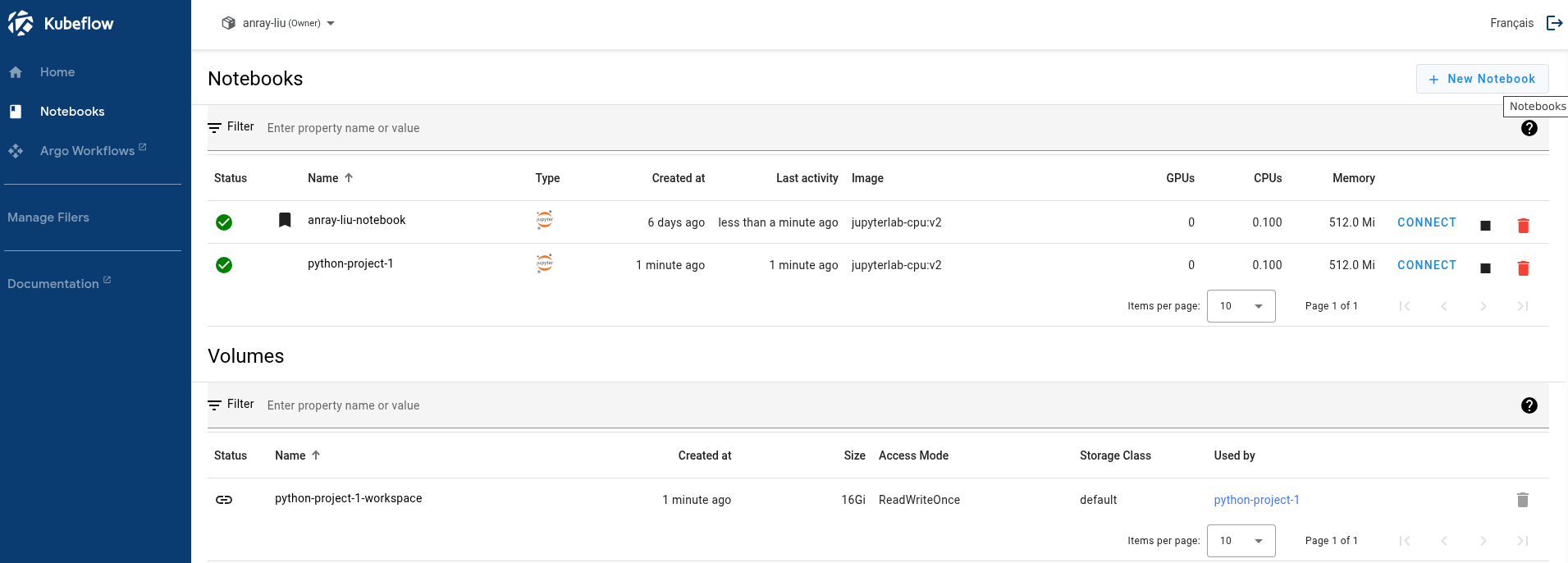
ORIGINALLY Prepared for statistics canada

<https://github.com/StatCan/volume-cleaner>

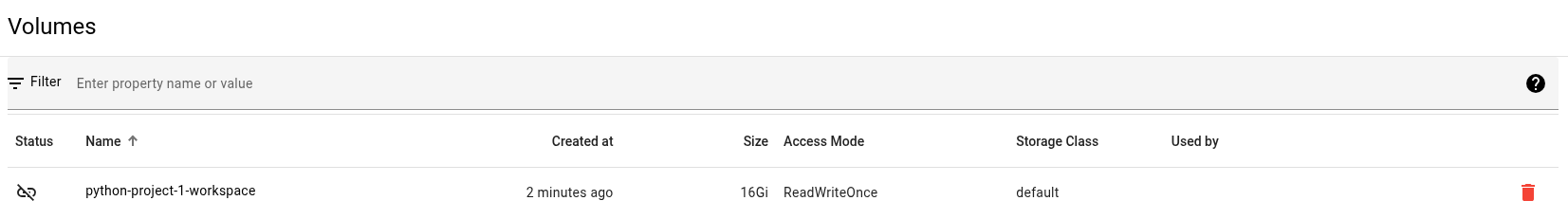
# Motivation

This project was designed to integrate with Kubeflow Notebooks. Kubeflow Notebooks are a Kubeflow component where users can create individual workspaces called Notebooks (Figure 1). To persist their work, users can attach volumes to their notebooks. These volumes are not automatically deleted when a notebook is deleted, and users are free to move or reuse them. As a result, over time, users tend to amass several unused volumes (Figure 2). These volumes sit on the cloud and result in unnecessary costs. The purpose of this project is to design an automatic system that detects these unused volumes and safely removes them, reducing unnecessary expenditure.

Despite being originally designed for Statistics Canada, this project strongly values open-source practices. The codebase is available for public use on the [Github repository](https://github.com/StatCan/volume-cleaner), and the entire development process is documented on the [issues](https://github.com/StatCan/volume-cleaner/issues) page. The volume cleaner was designed to have as little coupling as possible, so it can easily integrate into projects outside Statistics Canada. Using open-source and being open-source were important development philosophies.



*Figure 1: Kubeflow Notebooks Web Interface*



*Figure 2: Unattached Volume*

# Requirements

The project recognizes the following functional requirements (FR) and non-functional requirements (NFR):

1. Functional Requirement: Find unattached volumes.

Description: Volumes from namespaces in the cluster that have no associated notebook (“unattached volumes”) should automatically be discovered on a configurable basis using the built-in Unix cron scheduling from Kubernetes Cron Jobs. The namespaces to search should also be configurable based on labels.

1. Functional Requirement: Track unattached duration.

Description: The duration of which a volume has been continuously unattached should be tracked, and if it is longer than the specified grace period, it becomes “stale”. If a volume is reattached, its tracked time should reset.

1. Functional Requirement: Email users before deletion.

Description: Users should be emailed at configurable times before their volumes turn stale and are deleted.

1. Functional Requirement: Delete stale volumes and all associated resources.

Description: Stale volumes should be automatically deleted from the cloud. Proper cleanup should be done to ensure that all associated resources are deleted.

1. Functional Requirement: Run in a dry-run mode.

Description: In a toggleable dry-run mode, the volume cleaner should operate as usual, except that no resources will be deleted. This allows developers to safely monitor and test the volume cleaner without impacting the cluster.

1. Non-functional Requirement: Set a grace period before a volume turns stale.

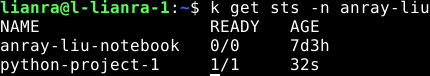
Description: Administrators should be able to configure the volume cleaner to specify the number of days a volume must be continuously unattached before it becomes stale.

1. Non-functional Requirement: Set intervals for user email notifications.

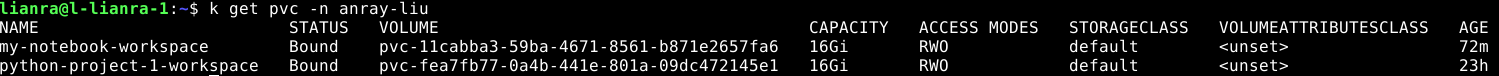
Description: Administrators should be able to configure a list of notification times. Whenever a volume is deleted in any of the given days, an email notification will be sent to the volume owner. For example, if the notification times are 30, 10, and 1, then an email will be sent to the owner 30 days before deletion, 10 days before deletion, and the day before deletion.

# Technical context

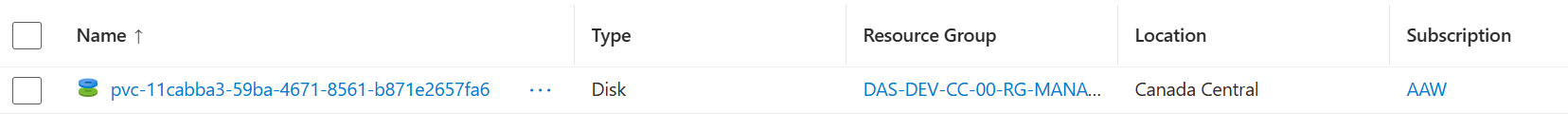
Kubernetes is the main backend component for Kubeflow Notebooks. Azure Kubernetes Service (AKS) is a cloud-based distribution of Kubernetes used by Statistics Canada. Users create notebooks and volumes through the Kubeflow user interface. When this happens, a Kubernetes Stateful Set (STS) is created to represent the notebook (Figure 3). When a notebook is activated, Kubernetes Pods are started under the STS. For each attached volume, a Kubernetes Persistent Volume Claim (PVC) and a Kubernetes Persistent Volume (PV) are created (Figure 4). A disk is then provisioned on Azure Disk Storage (Statistics Canada-specific, other projects may use other storage interfaces), which will store the user’s data (Figure 5). The disk is tied to the PV, which is tied to the PVC, which is tied to the STS. When a volume is deleted, it is important to ensure that this entire chain of resources is deleted to prevent orphaned resources. Neither the PV nor the PVC store the user’s data. They are simply managerial components for Kubernetes. For this project, Kubernetes Storage Classes do not have to be considered.



*Figure 3: STS’s Created for User Notebooks*

**

*Figure 4: PVCs and Bound PVs Created for User Volumes*

**

*Figure 5: Azure Disk Provisioned for a PV*

When a user deletes a notebook, its associated STS is automatically deleted. The rest of the resources are not affected. This is expected because the volume chain can later be reattached to another STS (notebook). When a user deletes a volume, the PVC is deleted, and the deletion propagates to the others until everything is cleaned up. After extensive testing that can be found at [#5](https://github.com/StatCan/volume-cleaner/issues/5), it was discovered that this behaviour is replicated when the PVC is manually deleted from the backend. Therefore, to ensure that the entire chain gets deleted, it is enough for the volume cleaner to only delete the PVC.

# Implementation

For the full discussion on architectural design, refer to [#26](https://github.com/StatCan/volume-cleaner/issues/26).

The volume cleaner consists of two parts. Both parts run on the AKS cluster. The first part is a controller that constantly monitors the cluster. This part is deployed as a Kubernetes Deployment. The controller watches for STS creations and deletions. Whenever an STS is deleted, each associated PVC receives a label with the current timestamp. The timestamp will later be used to determine how long the volume has been unattached. Whenever an STS is created, each associated PVC has its label removed.

One consideration is that if the controller goes down for any amount of time, it will not know what happened during the downtime. One edge case exists where while the controller is down, if a user creates a new volume and detaches it, that volume will never have an associated STS and as a result, will never get watched and cleaned up (see solution in section below). An even trickier edge case is when a user attaches and detaches an existing volume, because the controller would not know how to reset the label when it starts back up. This results in the eventual undesired deletion of the volume.

The second part is a Kubernetes Cronjob (CJ). The CJ periodically generates job pods that scan each of the PVCs. If the difference between the current timestamp and the PVC’s label is greater than the configured grace period, the PVC is stale and will be deleted. The deletion of a PVC will then trigger the deletion of its bound PV and Azure Disk. If the volume cleaner is running in dry mode, the PVC will not be deleted, and a description will be logged instead.

When the job checks the timestamps, it can also send an email notification to the volume owner, depending on the day. This list of days is configurable by administrators. If for example, the list consists of 30, 10, and 1, then every volume owner would receive an email warning 30 days before deletion, 10 days before deletion, and the day before deletion. Email is sent using third-party APIs. For Statistics Canada, this is done through an in-house service.

Go was chosen to be the programming language for several reasons, including being quick to write, easy to read, performant, and keeping align with many other Kubernetes services being written in Go. There is also a [Go client](https://github.com/kubernetes/client-go) package to access the cluster backend.

A diagram of a computer process

Description automatically generated

*Figure 6: Interaction between Volume Cleaner and Kubernetes Resources*

# Edge cases

The current implementation has the following edge cases:

1. Description: The controller is down. A user creates a new volume and detaches it. This volume is never tracked when the controller starts again because there is no STS to watch. Unattached volume is never deleted.

Solution: On startup, the controller will scan all PVCs and all STS’s. A cross comparison will be made to find all unattached PVCs. Assign these PVCs labels.

1. The controller is down. A user attaches an existing volume and detaches it. The label on this volume will not be reset because the controller would not know about the reattachment. After the grace period, this volume will be accidentally deleted.

Solution: Due to the complicated nature of implementing reliable logging during controller downtimes, it was decided to ignore this edge case for now. This edge case happens very rarely, and repeated email warnings should prevent most accidental deletions.

# usage

All the required manifests to deploy the volume cleaner can be found in the [manifests](https://github.com/StatCan/volume-cleaner/tree/main/manifests) folder. The most important files are controller\_deployment.yaml, scheduler\_job.yaml, and their respective configs and secrets. Other accompanying manifests, such as rbac.yaml, netpol.yaml, and serviceaccount.yaml are specific to a project’s network infrastructure. The manifests for the controller and scheduler both pull a docker image from a container registry. Therefore, there must be a system in place to build the go binaries, build the dockerfiles, and then push it to a container registry. Refer to the integration tests in the [GitHub actions workflow](https://github.com/StatCan/volume-cleaner/blob/main/.github/workflows/ci.yaml) for an example.

# performance

No issues to report so far. More detail will be added soon.

# Dependencies

The following list of Go packages need to be updated and maintained:

1. [kubernetes/client-go](https://github.com/kubernetes/client-go)
2. [kubernetes/apimachinery](https://github.com/kubernetes/apimachinery)
3. [kubernetes/api](https://github.com/kubernetes/api)
4. [github.com/stretchr/testify](https://github.com/stretchr/testify)

The volume cleaner needs no other external dependencies.