VMware Advanced Customer Engagements (ACE) Team

Quickstart Guide for Tanzu Kubernetes Grid Integrated (TKGI) Cluster Backup and Restore

June 2020

Table of Contents

Introduction

This document is a quickstart guide for backing up a Tanzu Kubernetes Grid Integrated (TKGI, formerly known as Enterprise PKS) Kubernetes cluster and restoring it. This document will provide details on Valero backup software, installing Velero, and backing up an existing cluster, and restoring to the same or another target cluster. The cluster backup will include all Kubernetes (K8) resources as well as persistent volumes.

Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns and more.

Kubernetes thus allows us to run a containerized application at scale. Running multiple replicas of the application ensures its high availability. Although Kubernetes provides HA with replicas, and zero downtime deployments, as with all platforms, we could face situations wherein a cluster can go into an unrecoverable state.

Kubernetes allows us to have a zero-downtime deployment, yet service interrupting events are inevitable and can occur at any time. Your network can go down, your latest application push can introduce a critical bug, or in the rarest case, you might even have to face a natural disaster.

We will hence require setting up a backup and recovery process to go back to the previously known stable state. Also, this process is useful to migrate workloads/cluster resources from one cluster to another etc.

Velero

Running on Kubernetes clusters or on VMs, Velero gives you tools to back up and restore your Kubernetes cluster resources and persistent volumes. You can run Velero in Kubernetes clusters from a cloud provider or on-premises. Velero lets you:

- Take backups of your cluster and restore in case of loss.
- Migrate cluster resources to other clusters.
- Replicate your production cluster to development and test clusters.

Velero consists of:

- A server that runs on your cluster
- A command-line client that runs locally

Disaster Recovery

If you periodically back up your cluster's resources, you are able to return to a previous state in case of some unexpected mishap, such as a service outage.

Cluster Migration

Velero can help you port your resources from one cluster to another, as long as you point each Velero instance to the same cloud object storage location.

Backup Reference

It is possible to exclude individual items from being backed up, even if they match the resource/namespace/label selectors defined in the backup spec.

Restore Reference

Velero can restore resources into a different namespace than the one they were backed up from.

How it Works

On-demand backups

- Uploads a tarball of copied Kubernetes objects into cloud object storage.
- Calls the cloud provider API to make disk snapshots of persistent volumes, if specified.

Scheduled backups

- The **schedule** operation allows you to back up your data at recurring intervals

Restores

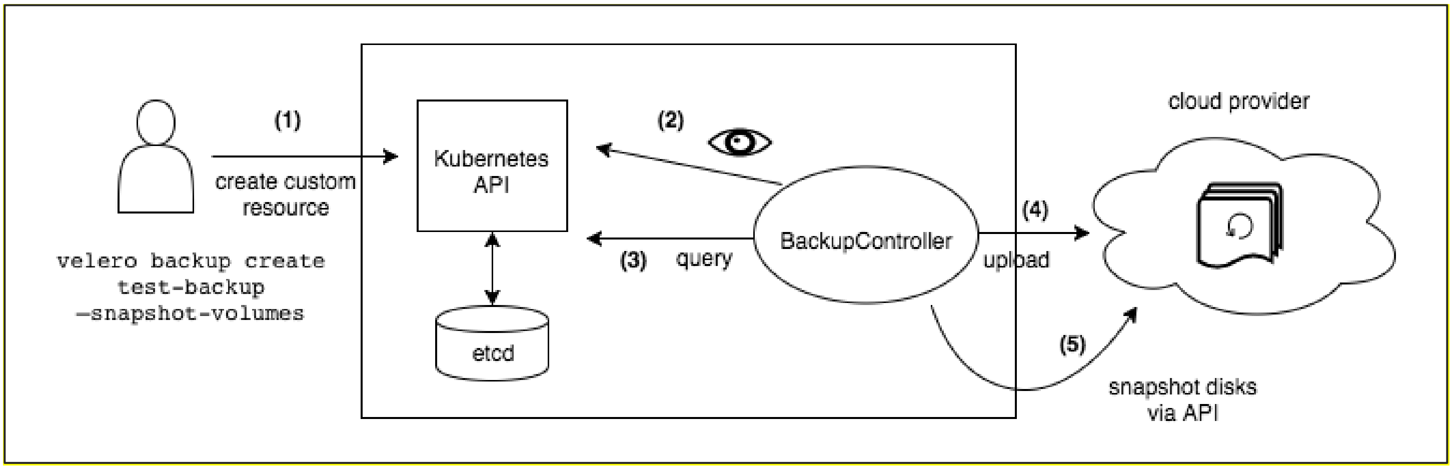
The **restore** operation allows you to restore all of the objects and persistent volumes from a previously created backup. You can also restore only a filtered subset of objects and persistent volumes.

Backup workflow
When you run Velero backup

- The Velero client makes a call to the Kubernetes API server to create a Backup object.
- The BackupController notices the new Backup object and performs validation.
- The BackupController begins the backup process. It collects the data to back up by querying the API server for resources.

The BackupController makes a call to the object storage service -- for example, AWS S3 -- to upload the backup

By default, Velero backup create makes disk snapshots of any persistent volumes. You can adjust the snapshots by specifying additional flags. Run Velero backup create --help to see available flags. Snapshots can be disabled with the option --snapshot-volumes=false.



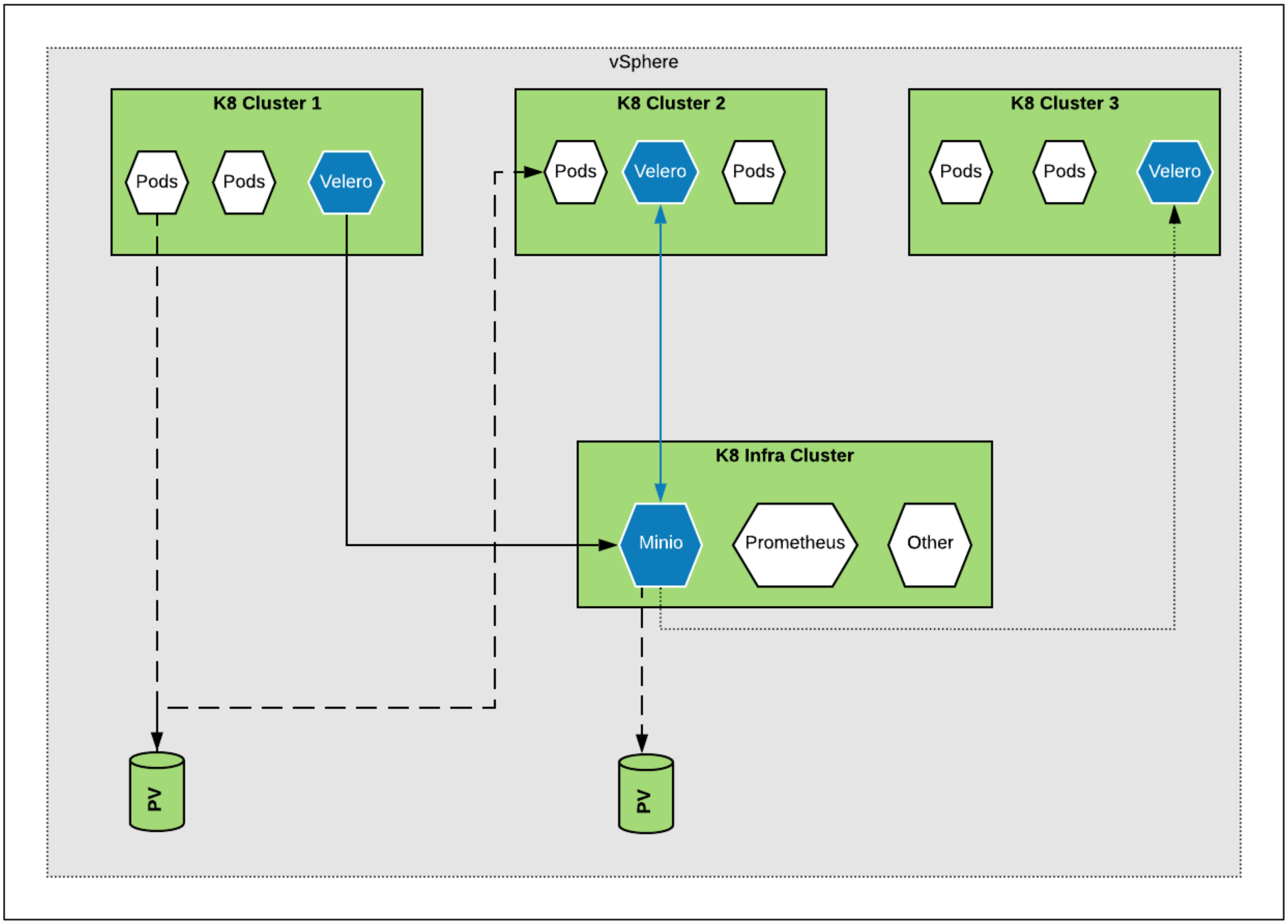
You can run Velero in Kubernetes clusters on a cloud provider or on-premises. For detailed information, see Compatible Storage Providers. Each Velero operation -- on-demand backup, scheduled backup, restore -- is a custom resource, defined with a Kubernetes Custom Resource Definition (CRD) and stored in etcd. Velero also includes controllers that process the custom resources to perform backups, restores, and all related operations. You can back up or restore all objects in your cluster, or you can filter objects by type, namespace, and/or label.

Restic inherently is a file-based backup. Currently, on a vSphere environment Velero uses Restic to backup Kubernetes Persistent Volumes (PV's) by taking the backup of all the files.

For more information go to https://velero.io

Use-case

For migrations or disaster recovery or maintenance, backup a Kubernetes cluster and restore its resources from a backup to a target cluster.



Assumptions

The following assumptions are made in the guide:

- TKGI is deployed
- The infrastructure team has setup 3 K8s clusters , the source cluster (where Velero backup is made from) , the target (where the Velero backup is restored to) cluster and an infra cluster where all infrastructure applications like Minio and Prometheus will be running .

  NOTE: This is not a hard and fast rule, Minio can run on any cluster, including the source and the target cluster, it could also be run as a standalone vm. For more information on minio visit *https://docs.min.io/*

- The Minio backup endpoint is accessible from both the source and target clusters.
- A Linux/ubuntu machine is provisioned to install and use various software components
- The provisioned Linux/ubuntu machine meets the following
  - Can access all the 3 K8s clusters defined above
  - Has the appropriate kubectl cli installed
  - Has the appropriate pks cli installed
  - Has the latest version of *Helm* installed
- In this document, we will be using ci-cluster as our source cluster and my-cluster as our target cluster.
- The source K8s clusters have applications deployed
  - A sample application is planespotter (*https://github.com/CNA-Tech/PKS-Ninja/tree/master/LabGuides/DeployPlanespotter-DP6539*)

Setup Minio

We will be setting up Minio in the infra structure cluster. This is not a hard and fast rule, Minio can run on any cluster, including the source and the target cluster, it could also be run as a standalone vm.. We will be using the Bitnami official Minio images for K8. The steps below describe how to setup Minio in a K8 cluster using the bitnami distribution.

Step 1: ssh to the provisioned ubuntu vm.

Step 2: Get kube config for the infra cluster

```
pks get-kubeconfig infra-cluster -a <pks api> -u <user> -p <password> -k
E.g.
pks get-kubeconfig infra-cluster -a pks.corp.local -u riaz -p VMware1! -k
```

Step 3: Create as namespace to which minio can be deployed

```
kubectl create ns minio
```

Step 4: Add Bitnami helm repository

```
Helm repo add Bitnami https://charts.bitnami.com/bitnami
```

Step 5: Minio requires a backing store / K8s persistent volume. Create a storage-class on the infra cluster with the following storage class definition. Copy the contents of the file below to a file storage-class.yaml and create the storage class.

```
---
kind: StorageClass
```

```
apiVersion: storage.k8s.io/v1
metadata:
  name: minio-disk
provisioner: kubernetes.io/vsphere-volume
parameters:
    diskformat: thin
```

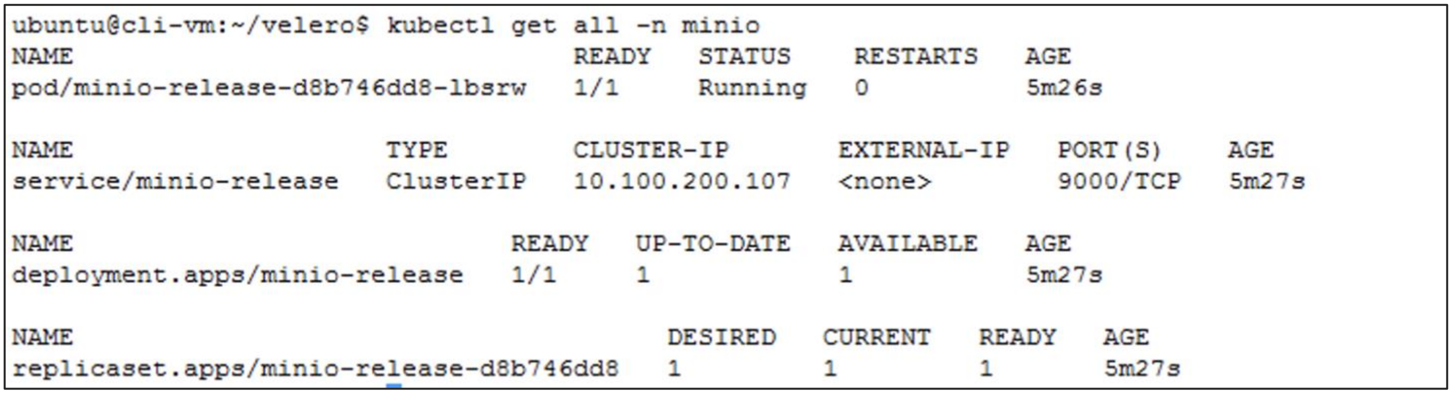
```
kubectl apply -f storage-class.yaml
```

Step 6: Deploy the Bitnami Minio release. This will create the necessary resources to run Minio within the minio namespace

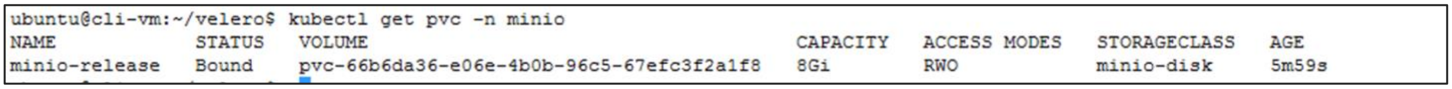
```
heml install minio-release -n minio \
  --set access.Key.password=minio \
  --set secretKey.password=minio123 \
  --set persistence.storageClass=minio-disk \
     bitnami/minio
```

Step 7: Check for all pods, deployments and services and make sure everything is created and the pods are running as expected. Also check if the PVC is created and bound

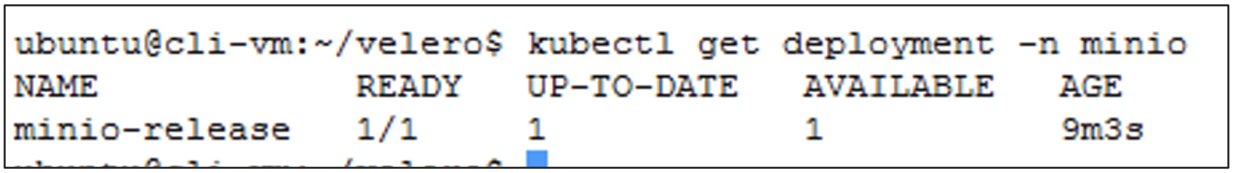
```
kubectl get all -n minio
kubectl get pvc -n minio
kubectl get deployment -n minio
```



```
ubuntu@cli-vm:~/velero$ kubectl get all -n minio
NAME                                  READY   STATUS    RESTARTS   AGE
pod/minio-release-d8b746dd8-lbsrw     1/1     Running   0          5m26s

NAME                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/minio-release      ClusterIP   10.100.200.107   <none>        9000/TCP   5m27s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/minio-release     1/1     1            1           5m27s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/minio-release-d8b746dd8     1         1         1       5m27s
```



```
ubuntu@cli-vm:~/velero$ kubectl get pvc -n minio
NAME            STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
minio-release   Bound    pvc-66b6da36-e06e-4b0b-96c5-67efc3f2a1f8   8Gi        RWO            minio-disk     5m59s
```



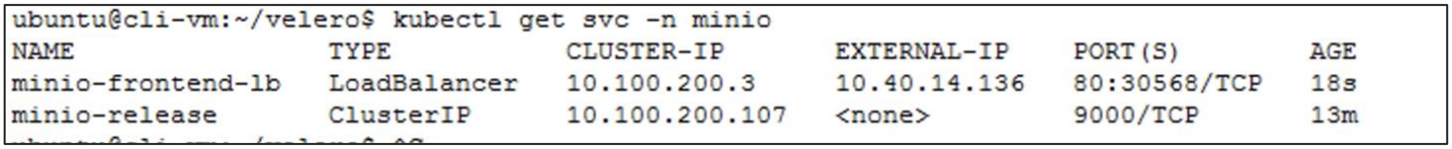
```
ubuntu@cli-vm:~/velero$ kubectl get deployment -n minio
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
minio-release   1/1     1            1           9m3s
```

Step 8: Expose the deployment as a loadbalancer. This will create any lb within NSXT as an ingress.

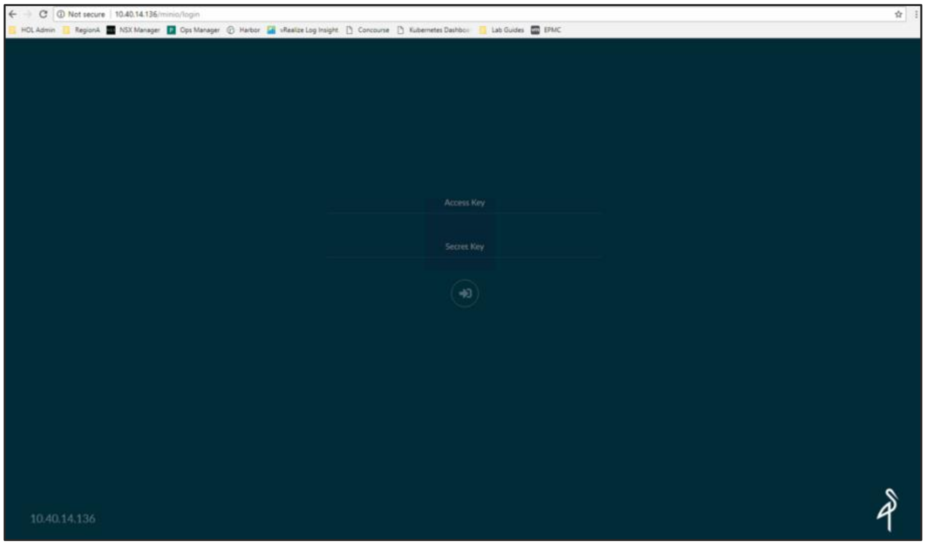
kubectl expose deployment minio-release --name=minio-frontend-lb --port=80 --target-port=9000 --type=LoadBalancer --namespace=minio

Step 9: Check the IP under the “External-IP” section, point your browser to the location <external-ip>. The Minio application should be accessible

kubectl get svc -n minio

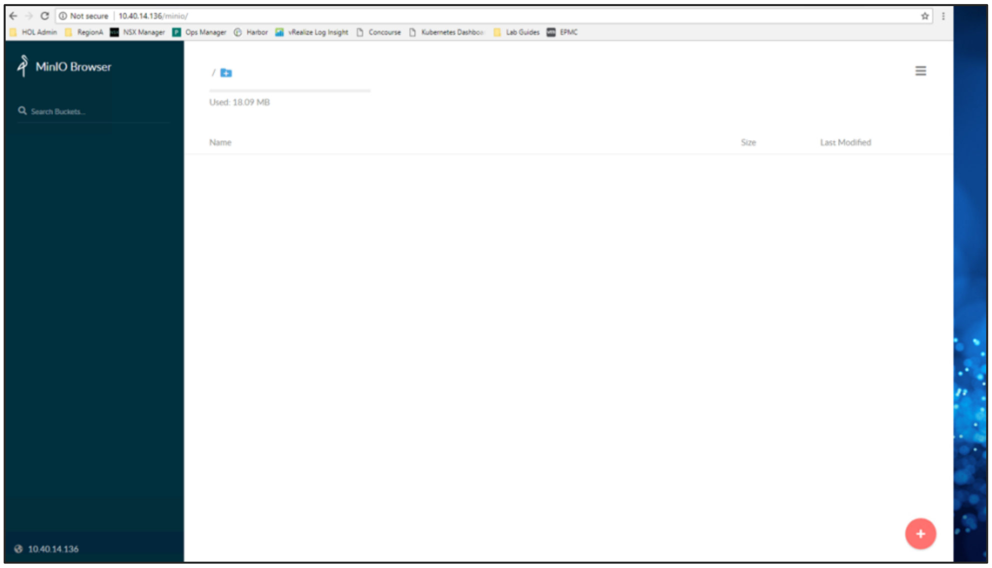


```
ubuntu@cli-vm:~/velero$ kubectl get svc -n minio
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)        AGE
minio-frontend-lb   LoadBalancer   10.100.200.3     10.40.14.136    80:30568/TCP   18s
minio-release       ClusterIP      10.100.200.107   <none>          9000/TCP       13m
```

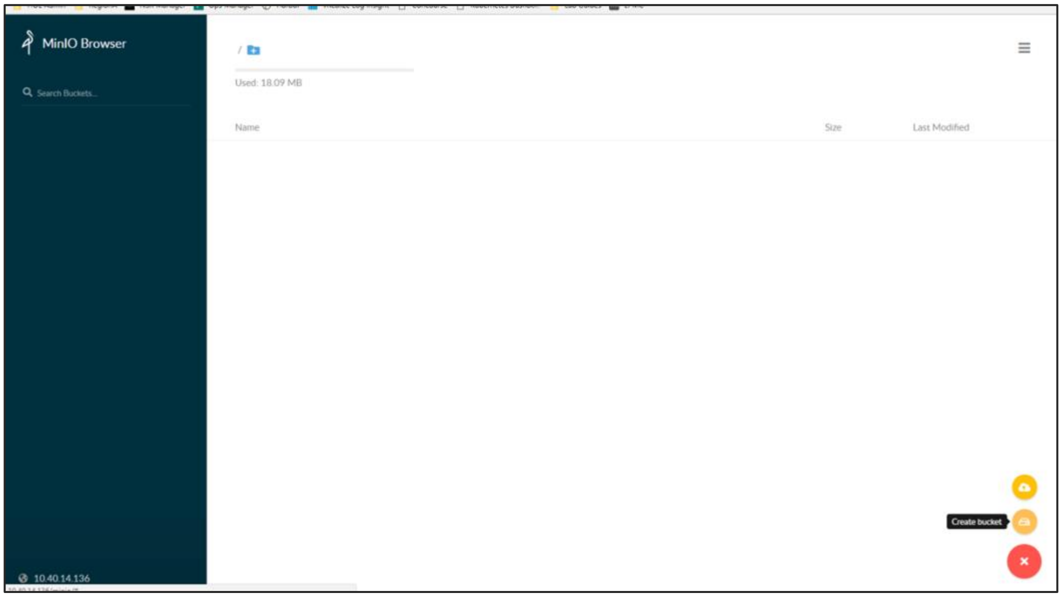


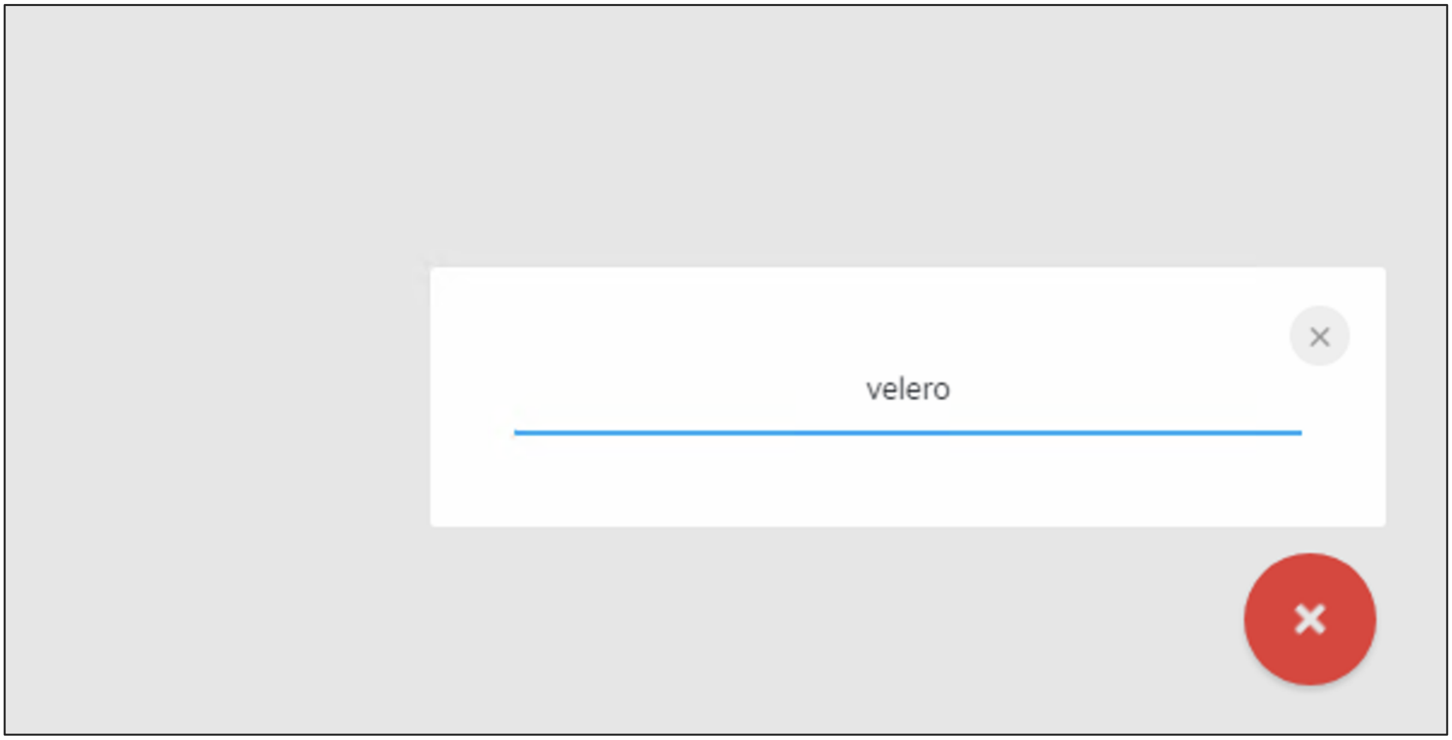
Not secure
10.40.14.136/minio/login
HOL Admin
RegionA
NSX Manager
Ops Manager
Harbor
vRealize Log Insight
Concourse
Kubernetes Dashboa
Lab Guides
EPMC
Access Key
Secret Key
10.40.14.136

Step 10: Login with the credentials used in step 6. - minio/minio123

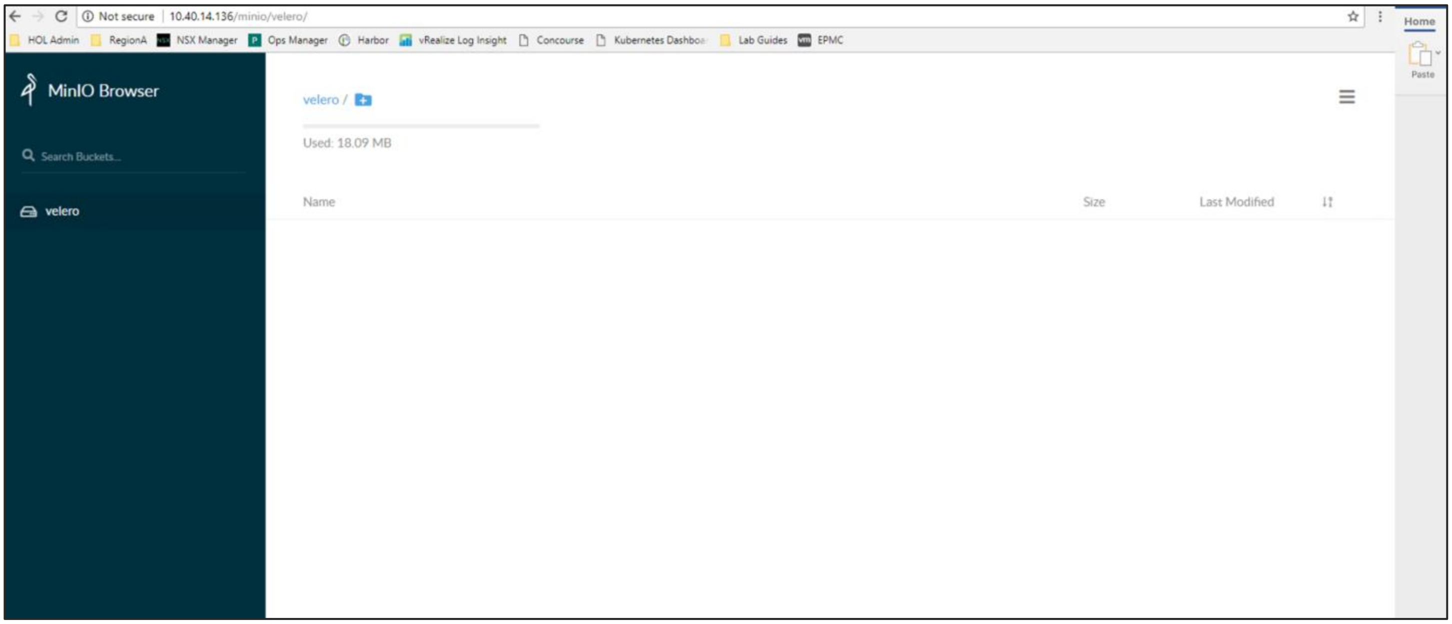


Step 11: Create a bucket called Velero. We will be using this bucket when we install Velero to the clusters in the following steps:





velero



10.40.14.136/minio/velero/

velero /

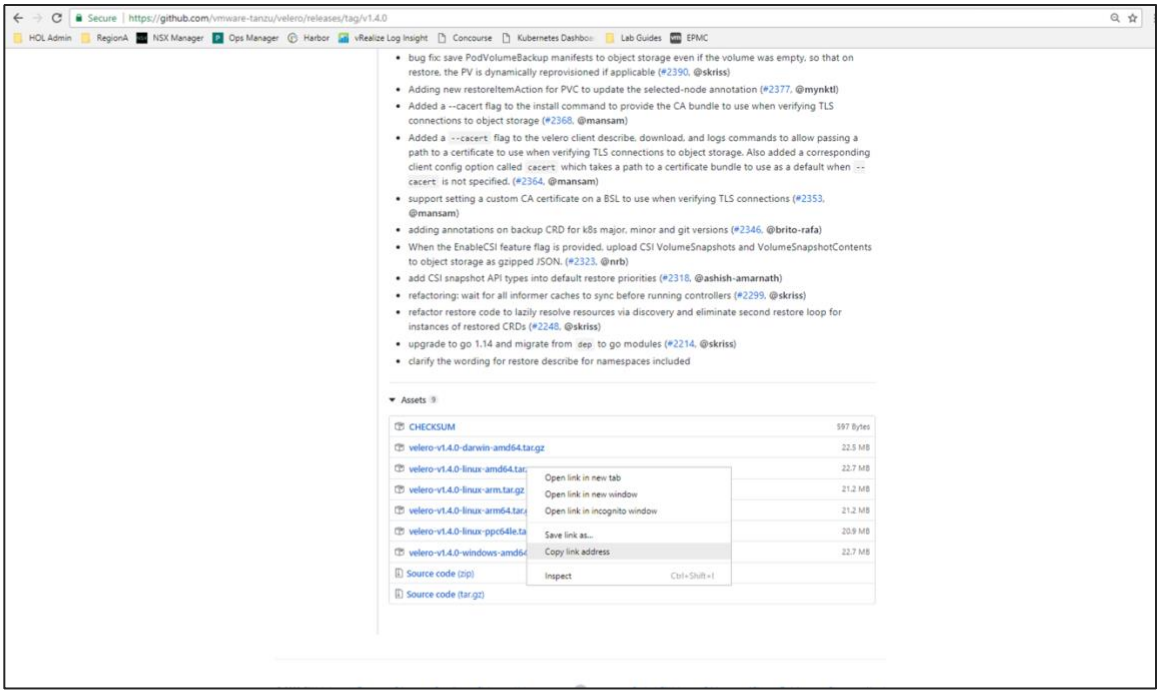
Used: 18.09 MB

Setup Velero

This section goes through the steps to download Velero to the provisioned Ubuntu vm and install Velero on both the source and target clusters.

Download Velero

Step 1: Navigate to the official page of Velero (*https://github.com/vmware-tanzu/velero/releases*) and copy the link for the target VM OS. (Eg. *https://github.com/vmware-tanzu/velero/releases/tag/v1.4.0*). At the bottom of the page the official releases are listed, Right clink on the release link 'Copy Link address'



- bug fix: save PodVolumeBackup manifests to object storage even if the volume was empty, so that on restore, the PV is dynamically reprovisioned if applicable (#2390, @skriss)
- Adding new restoreItemAction for PVC to update the selected-node annotation (#2377, @mynktl)
- Added a --cacert flag to the install command to provide the CA bundle to use when verifying TLS connections to object storage (#2368, @mansam)
- Added a `--cacert` flag to the velero client describe, download, and logs commands to allow passing a path to a certificate to use when verifying TLS connections to object storage. Also added a corresponding client config option called `cacert` which takes a path to a certificate bundle to use as a default when `--cacert` is not specified. (#2364, @mansam)
- support setting a custom CA certificate on a BSL to use when verifying TLS connections (#2353, @mansam)
- adding annotations on backup CRD for k8s major, minor and git versions (#2346, @brito-rafa)
- When the EnableCSI feature flag is provided, upload CSI VolumeSnapshots and VolumeSnapshotContents to object storage as gzipped JSON. (#2323, @nrb)
- add CSI snapshot API types into default restore priorities (#2318, @ashish-amarnath)
- refactoring: wait for all informer caches to sync before running controllers (#2299, @skriss)
- refactor restore code to lazily resolve resources via discovery and eliminate second restore loop for instances of restored CRDs (#2248, @skriss)
- upgrade to go 1.14 and migrate from `dep` to go modules (#2214, @skriss)
- clarify the wording for restore describe for namespaces included

Assets 9

CHECKSUM	597 Bytes
velero-v1.4.0-darwin-amd64.tar.gz	22.5 MB
velero-v1.4.0-linux-amd64.tar.	22.7 MB
velero-v1.4.0-linux-arm.tar.gz	21.2 MB
velero-v1.4.0-linux-arm64.tar.	21.2 MB
velero-v1.4.0-linux-ppc64le.ta	20.9 MB
velero-v1.4.0-windows-amd64	22.7 MB
Source code (zip)	
Source code (tar.gz)	

Step 2: ssh to the provisioned ubuntu vm.

Step 3: Download and uncompress the Velero distribution

```
mkdir velero
cd ~/velero
wget https://github.com/vmware-tanzu/velero/releases/download/v1.4.0/velero-v1.4.0-linux-amd64.tar.gz
```

```
tar xvf velero-v1.4.0-linux-amd64.tar.gz
```

Install Velero

This section describes the steps required to install Velero to both the source and target clusters. Any cluster from which a backup is taken or to which a backup is restored requires to have Velero deployed to it.

Source Cluster

Source cluster is the cluster from which a Velero backup will be taken from**.** As mentioned in the assumptions section we will be using the ci-cluster as our source cluster.

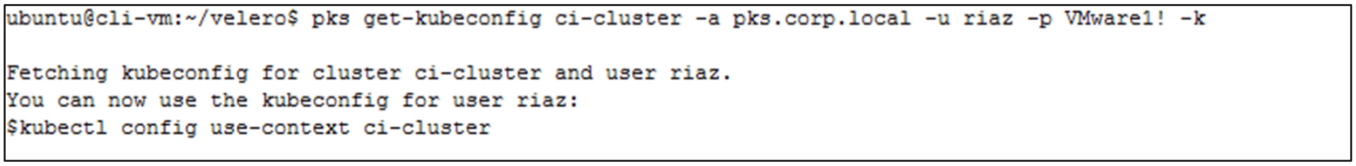
Step 1: ssh into the provisioned linux/ubuntu vm

Step 2: Get kube config for the source cluster

```
pks get-kubeconfig <source-cluster> -a <pks api> -u <user> -p <password> -k
```

E.g.

```
pks get-kubeconfig ci-cluster -a pks.corp.local -u riaz -p VMware1! -k
```



```
ubuntu@cli-vm:~/velero$ pks get-kubeconfig ci-cluster -a pks.corp.local -u riaz -p VMware1! -k

Fetching kubeconfig for cluster ci-cluster and user riaz.
You can now use the kubeconfig for user riaz:
$kubectl config use-context ci-cluster
```

Step 3: Create a velero namespace

```
kubectl create ns velero
```

Step 4: Change directory to the velero directory

```
cd ~/velero/velero-v1.4.0-linux-amd64
```

Step 5: Create a credentials file. Name it credentials This will contain the username and password used for Minio. The values would be the same as what was provided during the Minio setup.

```
[default]
        aws_access_key_id = minio
        aws_secret_access_key = minio123
```

Step 6: Set kubectl context to the source cluster

```
        kubectl config use-context <source-cluster>
        E.g.
        kubectl config use-context ci-cluster
```

Step 7: Install Velero to the source cluster.

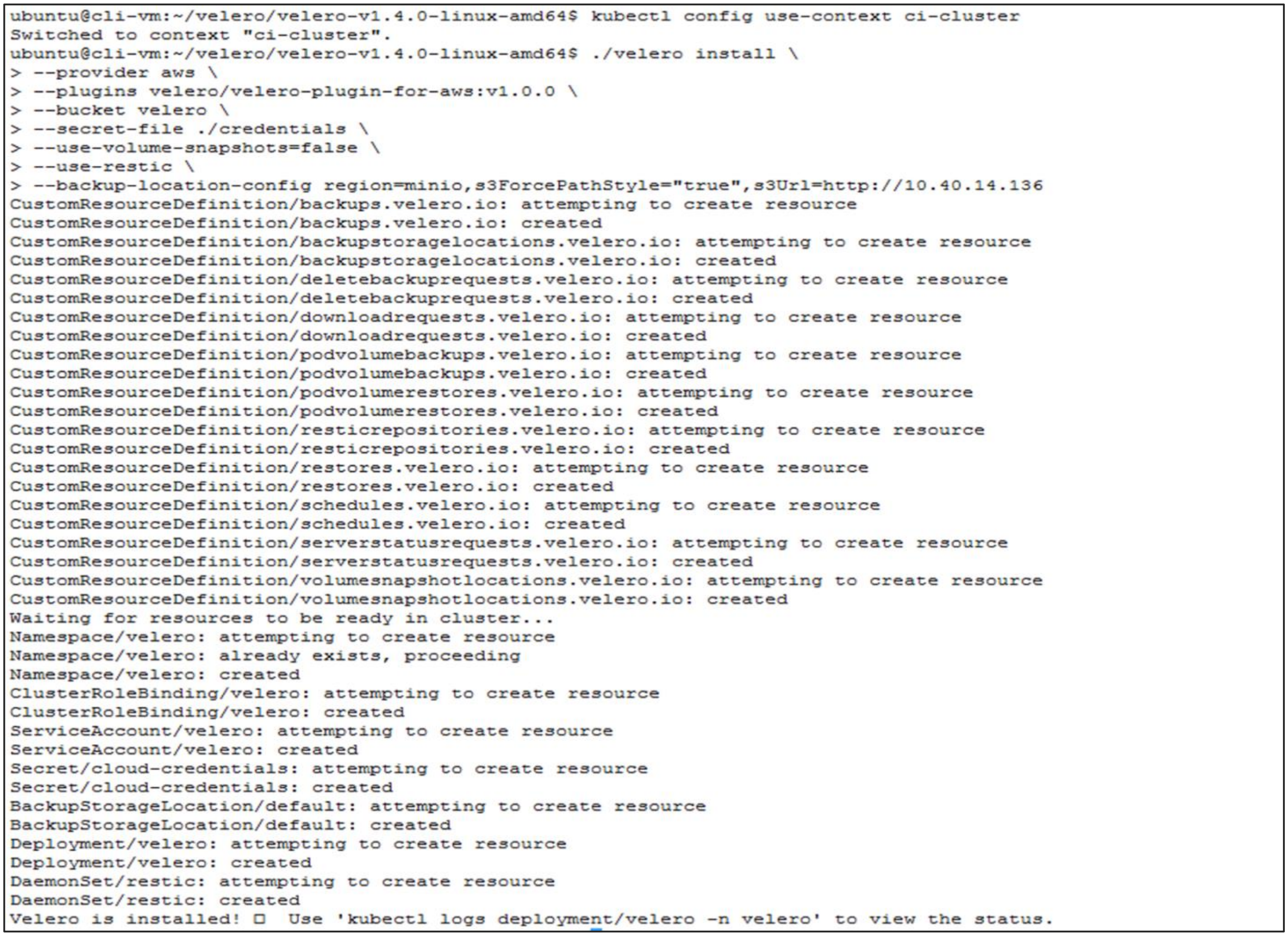
```
        ./velero install \
        --provider aws \
        --plugins velero/velero-plugin-for-aws:v1.0.0 \
        --bucket velero \
        --secret-file ./credentials \
        --use-volume-snapshots=false \
        --use-restic \
        --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://<externalip of minio>:<port>
```

Note: the secret file points to location of the file credentials file we created above

use restic to backup pv’s the s3Url points to the Minio that was setup earlier

E.g.

```
./velero install \
--provider aws \
--plugins velero/velero-plugin-for-aws:v1.0.0 \
--bucket velero \
--secret-file ./credentials \
--use-volume-snapshots=false \
--use-restic \
--backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://10.40.14.136
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl config use-context ci-cluster
Switched to context "ci-cluster".
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero install \
> --provider aws \
> --plugins velero/velero-plugin-for-aws:v1.0.0 \
> --bucket velero \
> --secret-file ./credentials \
> --use-volume-snapshots=false \
> --use-restic \
> --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://10.40.14.136
CustomResourceDefinition/backups.velero.io: attempting to create resource
CustomResourceDefinition/backups.velero.io: created
CustomResourceDefinition/backupstoragelocations.velero.io: attempting to create resource
CustomResourceDefinition/backupstoragelocations.velero.io: created
CustomResourceDefinition/deletebackuprequests.velero.io: attempting to create resource
CustomResourceDefinition/deletebackuprequests.velero.io: created
CustomResourceDefinition/downloadrequests.velero.io: attempting to create resource
CustomResourceDefinition/downloadrequests.velero.io: created
CustomResourceDefinition/podvolumebackups.velero.io: attempting to create resource
CustomResourceDefinition/podvolumebackups.velero.io: created
CustomResourceDefinition/podvolumerestores.velero.io: attempting to create resource
CustomResourceDefinition/podvolumerestores.velero.io: created
CustomResourceDefinition/resticrepositories.velero.io: attempting to create resource
CustomResourceDefinition/resticrepositories.velero.io: created
CustomResourceDefinition/restores.velero.io: attempting to create resource
CustomResourceDefinition/restores.velero.io: created
CustomResourceDefinition/schedules.velero.io: attempting to create resource
CustomResourceDefinition/schedules.velero.io: created
CustomResourceDefinition/serverstatusrequests.velero.io: attempting to create resource
CustomResourceDefinition/serverstatusrequests.velero.io: created
CustomResourceDefinition/volumesnapshotlocations.velero.io: attempting to create resource
CustomResourceDefinition/volumesnapshotlocations.velero.io: created
Waiting for resources to be ready in cluster...
Namespace/velero: attempting to create resource
Namespace/velero: already exists, proceeding
Namespace/velero: created
ClusterRoleBinding/velero: attempting to create resource
ClusterRoleBinding/velero: created
ServiceAccount/velero: attempting to create resource
ServiceAccount/velero: created
Secret/cloud-credentials: attempting to create resource
Secret/cloud-credentials: created
BackupStorageLocation/default: attempting to create resource
BackupStorageLocation/default: created
Deployment/velero: attempting to create resource
Deployment/velero: created
DaemonSet/restic: attempting to create resource
DaemonSet/restic: created
Velero is installed! □  Use 'kubectl logs deployment/velero -n velero' to view the status.
```

Step 8: Get status of pods in the velero namespace

kubectl get po -n velero

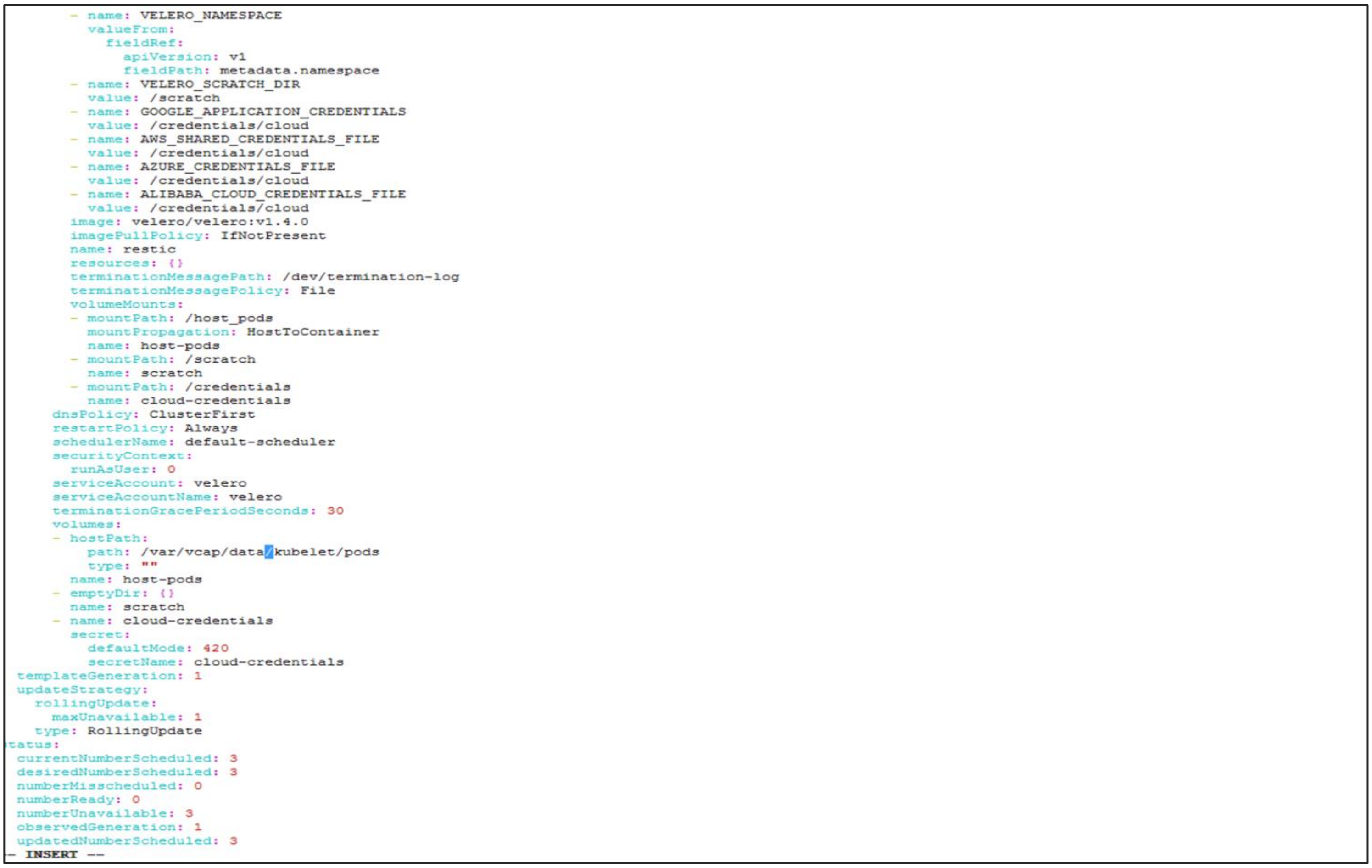
Step 9: If the Restic pods fail to startup we will need to edit the hostpath for the Restic pods

kubectl edit daemonset restic -n velero

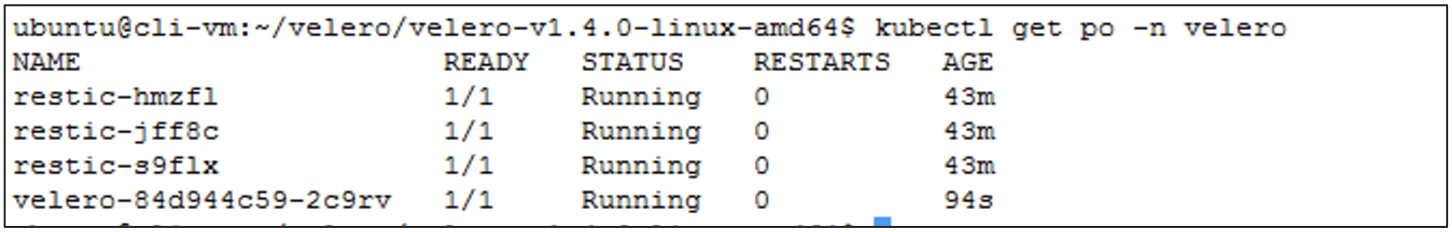
change hostPath from /var/lib/kubelet/pods to /var/vcap/data/kubelet/pods:

Which will look like below

```
          -hostPath:
               path: /var/vcap/data/kubelet/pods
```



```
        - name: VELERO_NAMESPACE
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
        - name: VELERO_SCRATCH_DIR
          value: /scratch
        - name: GOOGLE_APPLICATION_CREDENTIALS
          value: /credentials/cloud
        - name: AWS_SHARED_CREDENTIALS_FILE
          value: /credentials/cloud
        - name: AZURE_CREDENTIALS_FILE
          value: /credentials/cloud
        - name: ALIBABA_CLOUD_CREDENTIALS_FILE
          value: /credentials/cloud
        image: velero/velero:v1.4.0
        imagePullPolicy: IfNotPresent
        name: restic
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /host_pods
          mountPropagation: HostToContainer
          name: host-pods
        - mountPath: /scratch
          name: scratch
        - mountPath: /credentials
          name: cloud-credentials
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext:
        runAsUser: 0
      serviceAccount: velero
      serviceAccountName: velero
      terminationGracePeriodSeconds: 30
      volumes:
      - hostPath:
          path: /var/vcap/data/kubelet/pods
          type: ""
        name: host-pods
      - emptyDir: {}
        name: scratch
      - name: cloud-credentials
        secret:
          defaultMode: 420
          secretName: cloud-credentials
  templateGeneration: 1
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
tatus:
  currentNumberScheduled: 3
  desiredNumberScheduled: 3
  numberMisscheduled: 0
  numberReady: 0
  numberUnavailable: 3
  observedGeneration: 1
  updatedNumberScheduled: 3
- INSERT --
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get po -n velero
NAME                          READY   STATUS    RESTARTS   AGE
restic-hmzfl                  1/1     Running   0          43m
restic-jff8c                  1/1     Running   0          43m
restic-s9flx                  1/1     Running   0          43m
velero-84d944c59-2c9rv        1/1     Running   0          94s
```

Target Cluster

A target cluster is the cluster to which a Velero backup is to be restored**.** As mentioned in the assumptions section we will be using ‘my-cluster’ as our source cluster

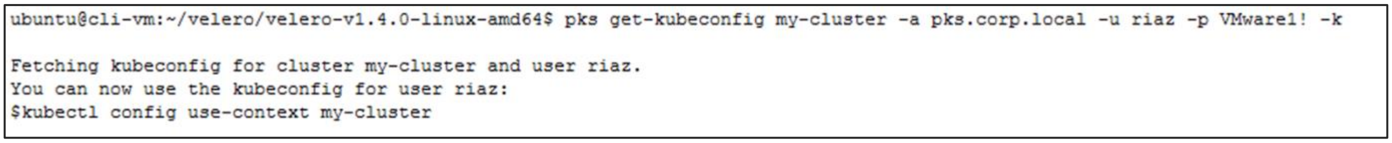
Step 1: ssh into the provisioned linux/ubuntu vm

Step 2: Get kube config for the target cluster

pks get-kubeconfig <source-cluster> -a <pks api> -u <user> -p <password> -k

E.g.

pks get-kubeconfig my-cluster -a pks.corp.local -u riaz -p VMware1! -k



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ pks get-kubeconfig my-cluster -a pks.corp.local -u riaz -p VMware1! -k

Fetching kubeconfig for cluster my-cluster and user riaz.
You can now use the kubeconfig for user riaz:
$kubectl config use-context my-cluster
```

Step 3: Create a velero namespace

kubectl create ns velero

Step 4: Change directory to the velero directory

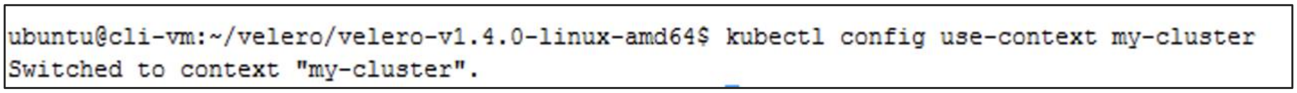
cd ~/velero/velero-v1.4.0-linux-amd64

Step 5: The credentials file should already exist; this would be the same as the one created for the source cluster

```
[default]
aws_access_key_id = minio
aws_secret_access_key = minio123
```

Step 7: Set kubectl context to the source cluster

```
kubectl config use-context <source-cluster>
E.g.
kubectl config use-context my-cluster
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl config use-context my-cluster
Switched to context "my-cluster".
```

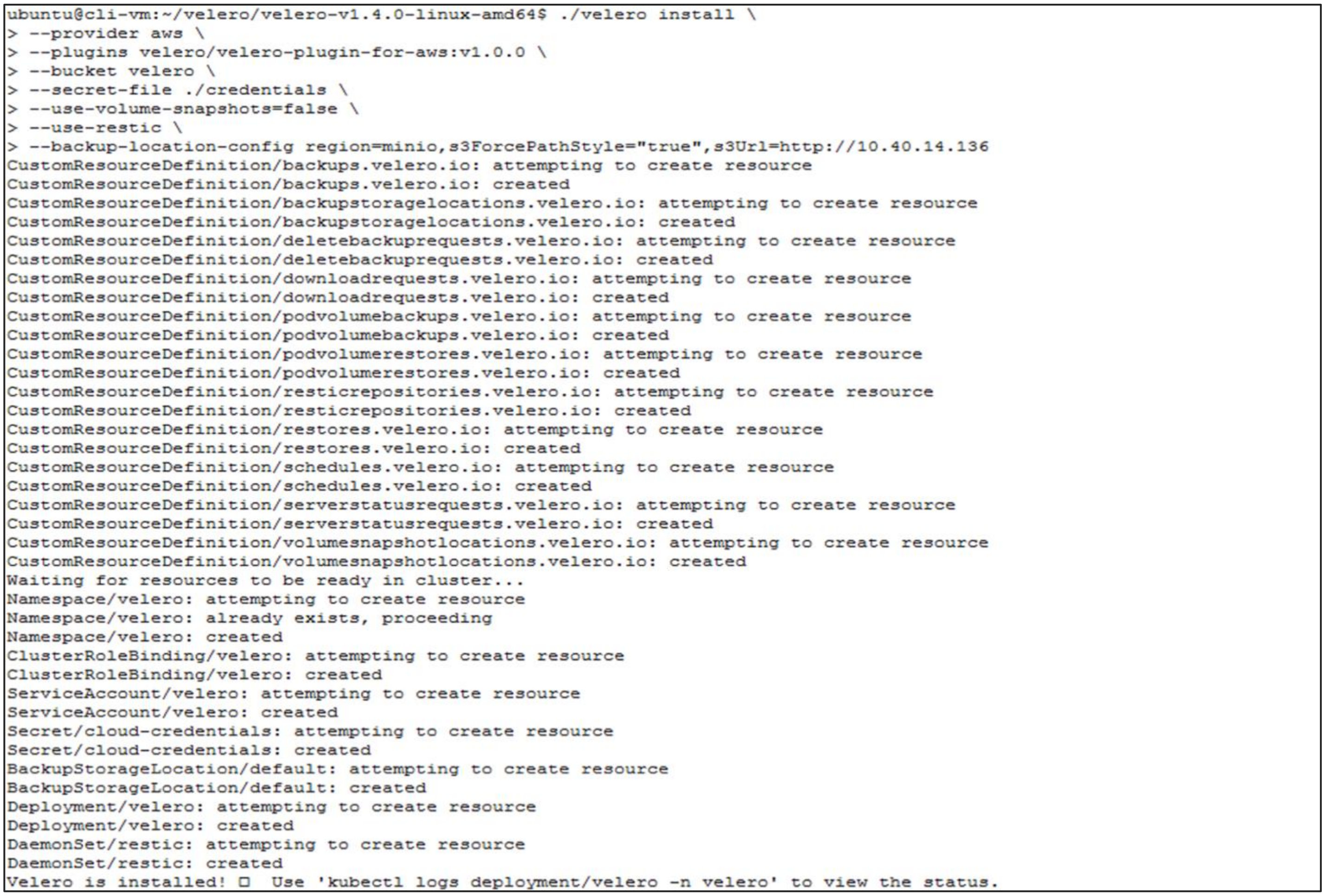
Step 6: Install Velero to the source cluster.

```
./velero install \
--provider aws \
--plugins velero/velero-plugin-for-aws:v1.0.0 \
--bucket velero \
--secret-file ./credentials \
--use-volume-snapshots=false \
--use-restic \
--backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://<externalip of minio>:<port>
```

Note: the secret file points to the file credentials file we created above use restic to backup pv’s the s3Url points to the Minio that was setup earlier

E.g.

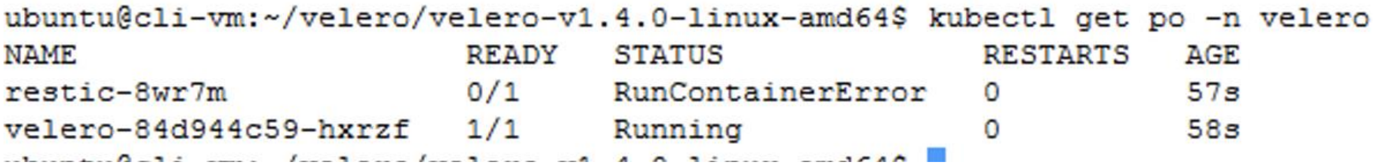
```
./velero install \
--provider aws \
--plugins velero/velero-plugin-for-aws:v1.0.0 \
--bucket velero \
--secret-file ./credentials \
--use-volume-snapshots=false \
--use-restic \
--backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://10.40.14.136
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero install \
> --provider aws \
> --plugins velero/velero-plugin-for-aws:v1.0.0 \
> --bucket velero \
> --secret-file ./credentials \
> --use-volume-snapshots=false \
> --use-restic \
> --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://10.40.14.136
CustomResourceDefinition/backups.velero.io: attempting to create resource
CustomResourceDefinition/backups.velero.io: created
CustomResourceDefinition/backupstoragelocations.velero.io: attempting to create resource
CustomResourceDefinition/backupstoragelocations.velero.io: created
CustomResourceDefinition/deletebackuprequests.velero.io: attempting to create resource
CustomResourceDefinition/deletebackuprequests.velero.io: created
CustomResourceDefinition/downloadrequests.velero.io: attempting to create resource
CustomResourceDefinition/downloadrequests.velero.io: created
CustomResourceDefinition/podvolumebackups.velero.io: attempting to create resource
CustomResourceDefinition/podvolumebackups.velero.io: created
CustomResourceDefinition/podvolumerestores.velero.io: attempting to create resource
CustomResourceDefinition/podvolumerestores.velero.io: created
CustomResourceDefinition/resticrepositories.velero.io: attempting to create resource
CustomResourceDefinition/resticrepositories.velero.io: created
CustomResourceDefinition/restores.velero.io: attempting to create resource
CustomResourceDefinition/restores.velero.io: created
CustomResourceDefinition/schedules.velero.io: attempting to create resource
CustomResourceDefinition/schedules.velero.io: created
CustomResourceDefinition/serverstatusrequests.velero.io: attempting to create resource
CustomResourceDefinition/serverstatusrequests.velero.io: created
CustomResourceDefinition/volumesnapshotlocations.velero.io: attempting to create resource
CustomResourceDefinition/volumesnapshotlocations.velero.io: created
Waiting for resources to be ready in cluster...
Namespace/velero: attempting to create resource
Namespace/velero: already exists, proceeding
Namespace/velero: created
ClusterRoleBinding/velero: attempting to create resource
ClusterRoleBinding/velero: created
ServiceAccount/velero: attempting to create resource
ServiceAccount/velero: created
Secret/cloud-credentials: attempting to create resource
Secret/cloud-credentials: created
BackupStorageLocation/default: attempting to create resource
BackupStorageLocation/default: created
Deployment/velero: attempting to create resource
Deployment/velero: created
DaemonSet/restic: attempting to create resource
DaemonSet/restic: created
Velero is installed! □  Use 'kubectl logs deployment/velero -n velero' to view the status.
```

Step 7: Get status of pods in the velero namespace

```
kubectl get po -n velero
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get po -n velero
NAME                        READY   STATUS              RESTARTS   AGE
restic-8wr7m                0/1     RunContainerError   0          57s
velero-84d944c59-hxrzf      1/1     Running             0          58s
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$
```

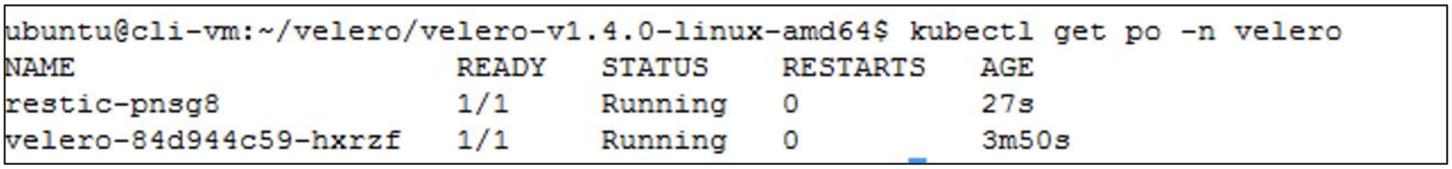
Step 8: If the restic pods fail to startup we will need to edit the hostpath for the restic pods

```
kubectl edit daemonset restic -n velero
```

change hostPath from /var/lib/kubelet/pods to /var/vcap/data/kubelet/pods:

Which will look like below

```
-hostPath:
    path: /var/vcap/data/kubelet/pods
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get po -n velero
NAME                        READY   STATUS    RESTARTS   AGE
restic-pnsg8                1/1     Running   0          27s
velero-84d944c59-hxrzf      1/1     Running   0          3m50s
```

Backup the Source Cluster

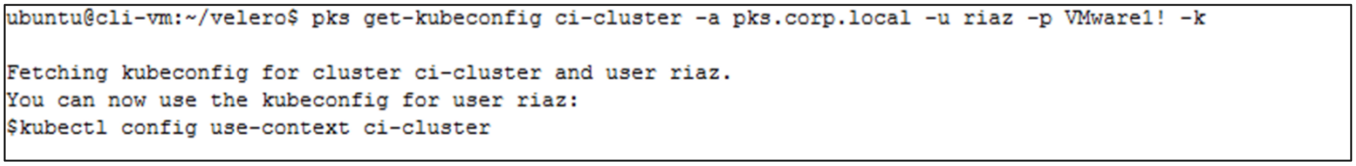
This section describes steps to backup a source cluster. The steps give an overview of backing up all the resources in a cluster as well as backing up just a namespace in a cluster

Step 1: Get kube config for the source cluster

pks get-kubeconfig <source-cluster> -a <pks api> -u <user> -p <password> -k

E.g.

pks get-kubeconfig ci-cluster -a pks.corp.local -u riaz -p VMware1! -k



```
ubuntu@cli-vm:~/velero$ pks get-kubeconfig ci-cluster -a pks.corp.local -u riaz -p VMware1! -k

Fetching kubeconfig for cluster ci-cluster and user riaz.
You can now use the kubeconfig for user riaz:
$kubectl config use-context ci-cluster
```

Step 2: Set kubectl context to the source cluster

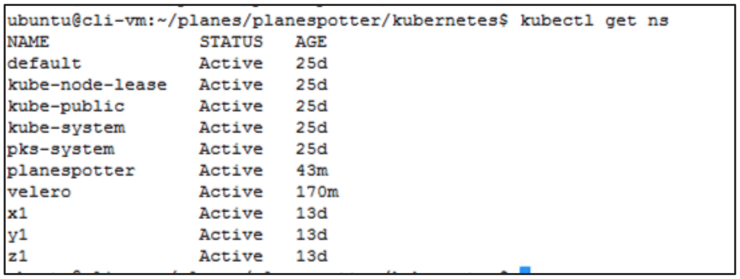
kubectl config use-context <source-cluster>

E.g.

kubectl config use-context ci-cluster

Step 3: Check all resources running on the source cluster

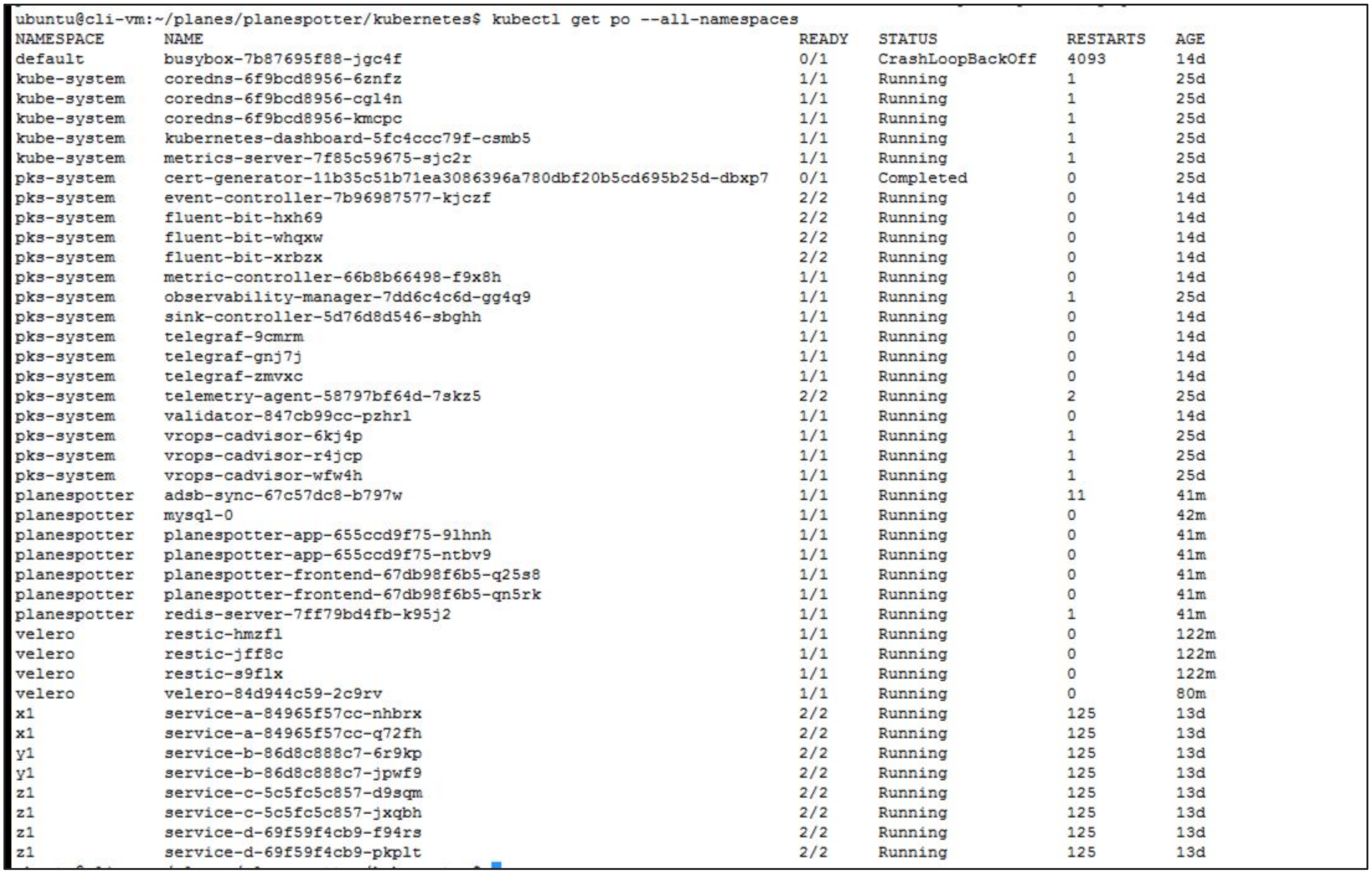
kubectl get ns



```
ubuntu@cli-vm:~/planes/planespotter/kubernetes$ kubectl get ns
NAME              STATUS   AGE
default           Active   25d
kube-node-lease   Active   25d
kube-public       Active   25d
kube-system       Active   25d
pks-system        Active   25d
planespotter      Active   43m
velero            Active   170m
x1                Active   13d
y1                Active   13d
z1                Active   13d
```

NOTE: apart from the default and system namespaces, planespotter, x1, y1 and z1 exist

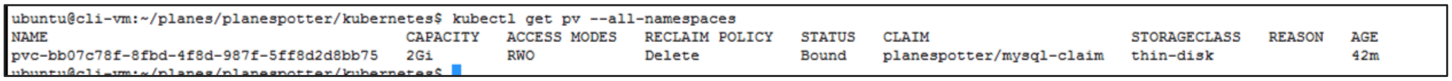
kubectl get po --all-namespaces



```
ubuntu@cli-vm:~/planes/planespotter/kubernetes$ kubectl get po --all-namespaces
NAMESPACE      NAME                                                             READY   STATUS             RESTARTS   AGE
default        busybox-7b87695f88-jgc4f                                         0/1     CrashLoopBackOff   4093       14d
kube-system    coredns-6f9bcd8956-6znfz                                         1/1     Running            1          25d
kube-system    coredns-6f9bcd8956-cgl4n                                         1/1     Running            1          25d
kube-system    coredns-6f9bcd8956-kmcpc                                         1/1     Running            1          25d
kube-system    kubernetes-dashboard-5fc4ccc79f-csmb5                            1/1     Running            1          25d
kube-system    metrics-server-7f85c59675-sjc2r                                  1/1     Running            1          25d
pks-system     cert-generator-11b35c51b71ea3086396a780dbf20b5cd695b25d-dbxp7    0/1     Completed          0          25d
pks-system     event-controller-7b96987577-kjczf                                2/2     Running            0          14d
pks-system     fluent-bit-hxh69                                                 2/2     Running            0          14d
pks-system     fluent-bit-whqxw                                                 2/2     Running            0          14d
pks-system     fluent-bit-xrbzx                                                 2/2     Running            0          14d
pks-system     metric-controller-66b8b66498-f9x8h                               1/1     Running            0          14d
pks-system     observability-manager-7dd6c4c6d-gg4q9                            1/1     Running            1          25d
pks-system     sink-controller-5d76d8d546-sbghh                                 1/1     Running            0          14d
pks-system     telegraf-9cmrm                                                   1/1     Running            0          14d
pks-system     telegraf-gnj7j                                                   1/1     Running            0          14d
pks-system     telegraf-zmvxc                                                   1/1     Running            0          14d
pks-system     telemetry-agent-58797bf64d-7skz5                                 2/2     Running            2          25d
pks-system     validator-847cb99cc-pzhrl                                        1/1     Running            0          14d
pks-system     vrops-cadvisor-6kj4p                                             1/1     Running            1          25d
pks-system     vrops-cadvisor-r4jcp                                             1/1     Running            1          25d
pks-system     vrops-cadvisor-wfw4h                                             1/1     Running            1          25d
planespotter   adsb-sync-67c57dc8-b797w                                         1/1     Running            11         41m
planespotter   mysql-0                                                          1/1     Running            0          42m
planespotter   planespotter-app-655ccd9f75-9lhnh                                1/1     Running            0          41m
planespotter   planespotter-app-655ccd9f75-ntbv9                                1/1     Running            0          41m
planespotter   planespotter-frontend-67db98f6b5-q25s8                           1/1     Running            0          41m
planespotter   planespotter-frontend-67db98f6b5-qn5rk                           1/1     Running            0          41m
planespotter   redis-server-7ff79bd4fb-k95j2                                    1/1     Running            1          41m
velero         restic-hmzfl                                                     1/1     Running            0          122m
velero         restic-jff8c                                                     1/1     Running            0          122m
velero         restic-s9flx                                                     1/1     Running            0          122m
velero         velero-84d944c59-2c9rv                                           1/1     Running            0          80m
x1             service-a-84965f57cc-nhbrx                                       2/2     Running            125        13d
x1             service-a-84965f57cc-q72fh                                       2/2     Running            125        13d
y1             service-b-86d8c888c7-6r9kp                                       2/2     Running            125        13d
y1             service-b-86d8c888c7-jpwf9                                       2/2     Running            125        13d
z1             service-c-5c5fc5c857-d9sqm                                       2/2     Running            125        13d
z1             service-c-5c5fc5c857-jxqbh                                       2/2     Running            125        13d
z1             service-d-69f59f4cb9-f94rs                                       2/2     Running            125        13d
z1             service-d-69f59f4cb9-pkplt                                       2/2     Running            125        13d
```

NOTE: Note the pods running in the default, planespotter and the x1, y1 and z1 namespaces

kubectl get pv --all-namespaces

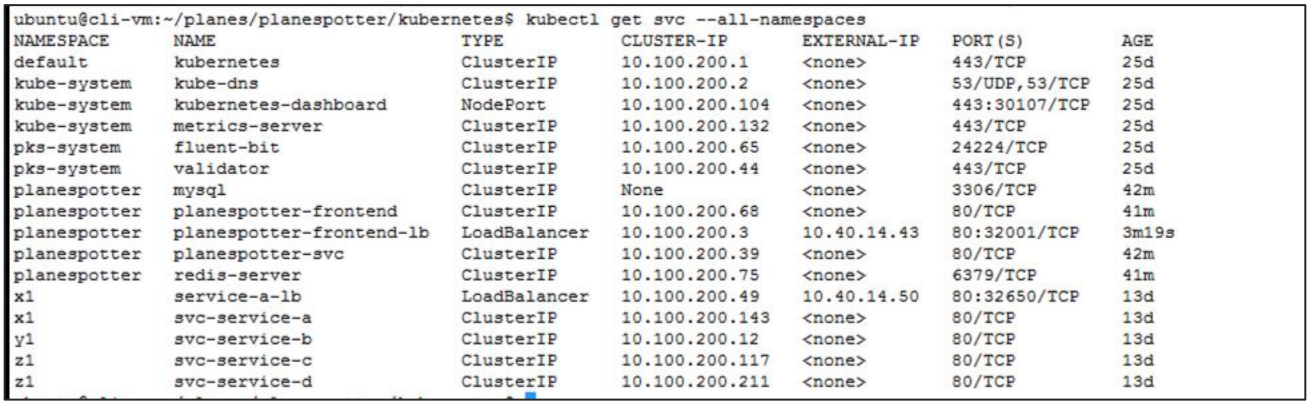


```
ubuntu@cli-vm:~/planes/planespotter/kubernetes$ kubectl get pv --all-namespaces
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                          STORAGECLASS   REASON   AGE
pvc-bb07c78f-8fbd-4f8d-987f-5ff8d2d8bb75   2Gi        RWO            Delete           Bound    planespotter/mysql-claim       thin-disk               42m
ubuntu@cli-vm:~/planes/planespotter/kubernetes$
```

NOTE: There is a PV in the planespotter namespace, which contains DB data for MySQL

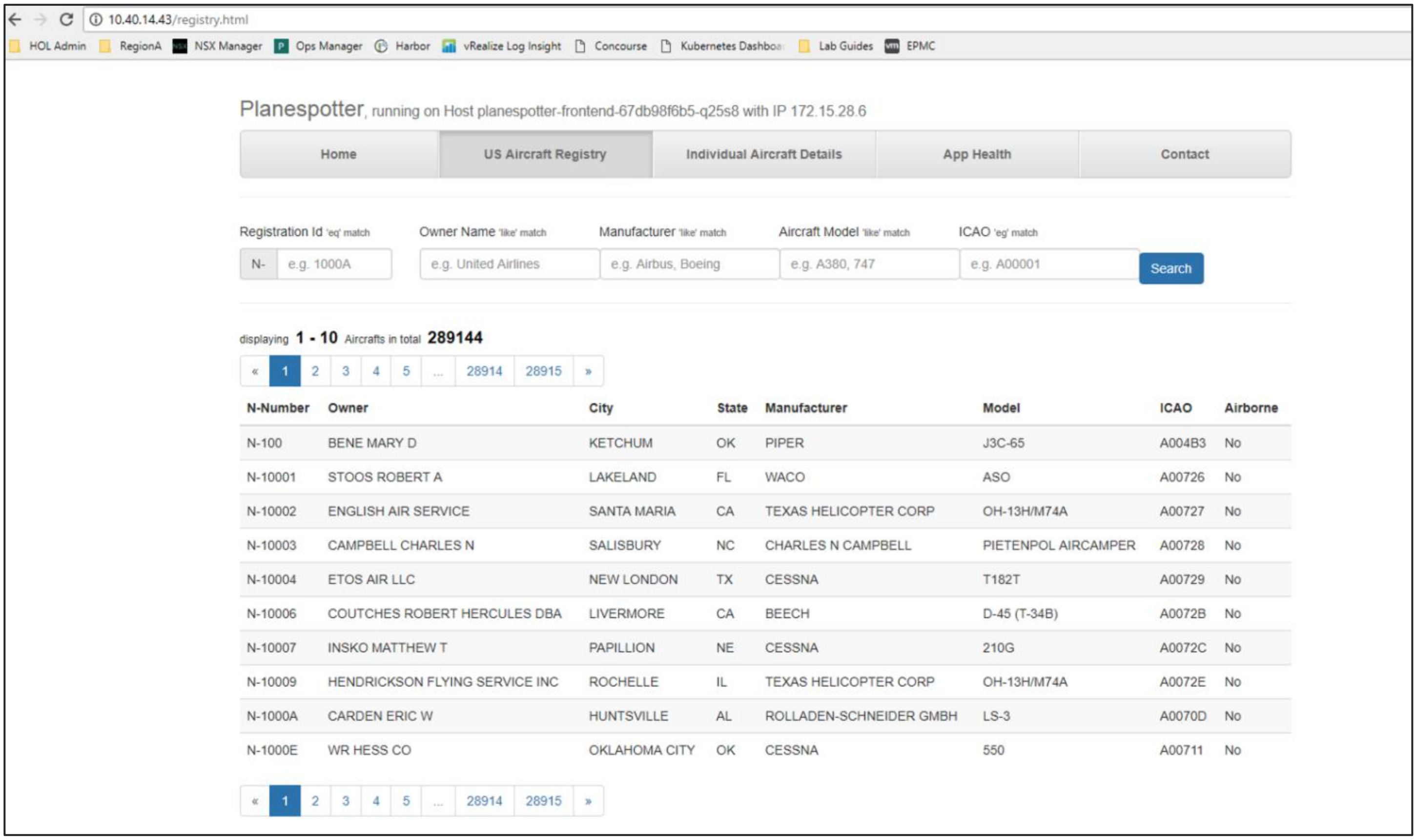
Step 4: Login to the planespotter and make sure everything is working and the DB data is being displayed

kubectl get svc --all-namespaces



```
ubuntu@cli-vm:~/planes/planespotter/kubernetes$ kubectl get svc --all-namespaces
NAMESPACE      NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
default        kubernetes                  ClusterIP      10.100.200.1     <none>        443/TCP          25d
kube-system    kube-dns                    ClusterIP      10.100.200.2     <none>        53/UDP,53/TCP    25d
kube-system    kubernetes-dashboard        NodePort       10.100.200.104   <none>        443:30107/TCP    25d
kube-system    metrics-server              ClusterIP      10.100.200.132   <none>        443/TCP          25d
pks-system     fluent-bit                  ClusterIP      10.100.200.65    <none>        24224/TCP        25d
pks-system     validator                   ClusterIP      10.100.200.44    <none>        443/TCP          25d
planespotter   mysql                       ClusterIP      None             <none>        3306/TCP         42m
planespotter   planespotter-frontend       ClusterIP      10.100.200.68    <none>        80/TCP           41m
planespotter   planespotter-frontend-lb    LoadBalancer   10.100.200.3     10.40.14.43   80:32001/TCP     3m19s
planespotter   planespotter-svc            ClusterIP      10.100.200.39    <none>        80/TCP           42m
planespotter   redis-server                ClusterIP      10.100.200.75    <none>        6379/TCP         41m
x1             service-a-lb                LoadBalancer   10.100.200.49    10.40.14.50   80:32650/TCP     13d
x1             svc-service-a               ClusterIP      10.100.200.143   <none>        80/TCP           13d
y1             svc-service-b               ClusterIP      10.100.200.12    <none>        80/TCP           13d
z1             svc-service-c               ClusterIP      10.100.200.117   <none>        80/TCP           13d
z1             svc-service-d               ClusterIP      10.100.200.211   <none>        80/TCP           13d
```

Point the browser to the external-ip of the planespotter-frontend-lb microservice



Planespotter, running on Host planespotter-frontend-67db98f6b5-q25s8 with IP 172.15.28.6

displaying 1 - 10 Aircrafts in total 289144

N-Number	Owner	City	State	Manufacturer	Model	ICAO	Airborne
N-100	BENE MARY D	KETCHUM	OK	PIPER	J3C-65	A004B3	No
N-10001	STOOS ROBERT A	LAKELAND	FL	WACO	ASO	A00726	No
N-10002	ENGLISH AIR SERVICE	SANTA MARIA	CA	TEXAS HELICOPTER CORP	OH-13H/M74A	A00727	No
N-10003	CAMPBELL CHARLES N	SALISBURY	NC	CHARLES N CAMPBELL	PIETENPOL AIRCAMPER	A00728	No
N-10004	ETOS AIR LLC	NEW LONDON	TX	CESSNA	T182T	A00729	No
N-10006	COUTCHES ROBERT HERCULES DBA	LIVERMORE	CA	BEECH	D-45 (T-34B)	A0072B	No
N-10007	INSKO MATTHEW T	PAPILLION	NE	CESSNA	210G	A0072C	No
N-10009	HENDRICKSON FLYING SERVICE INC	ROCHELLE	IL	TEXAS HELICOPTER CORP	OH-13H/M74A	A0072E	No
N-1000A	CARDEN ERIC W	HUNTSVILLE	AL	ROLLADEN-SCHNEIDER GMBH	LS-3	A0070D	No
N-1000E	WR HESS CO	OKLAHOMA CITY	OK	CESSNA	550	A00711	No

Step 5: For the planespotter app, mysql is the stateful pod that would need to be annotated. (Volumes from mysql_pod.yaml in the planespotter app). All stateful pods need to be annotated

Run the following to annotate each pod that contains a volume to back up

```
kubectl -n YOUR_POD_NAMESPACE annotate pod/YOUR_POD_NAME backup.velero.io/backup-volumes=YOUR_VOLUME_NAME_1,YOUR_VOLUME_NAME_2,...
```

STEP 6: To find the Volumes for the stateful pod, identify the stateful pod and describe it. For eg. in our example mysql-0 is the stateful pod

```
kubectl describe po mysql-0 -n planespotter
```

The volumes are

Volumes:

mysql-vol:

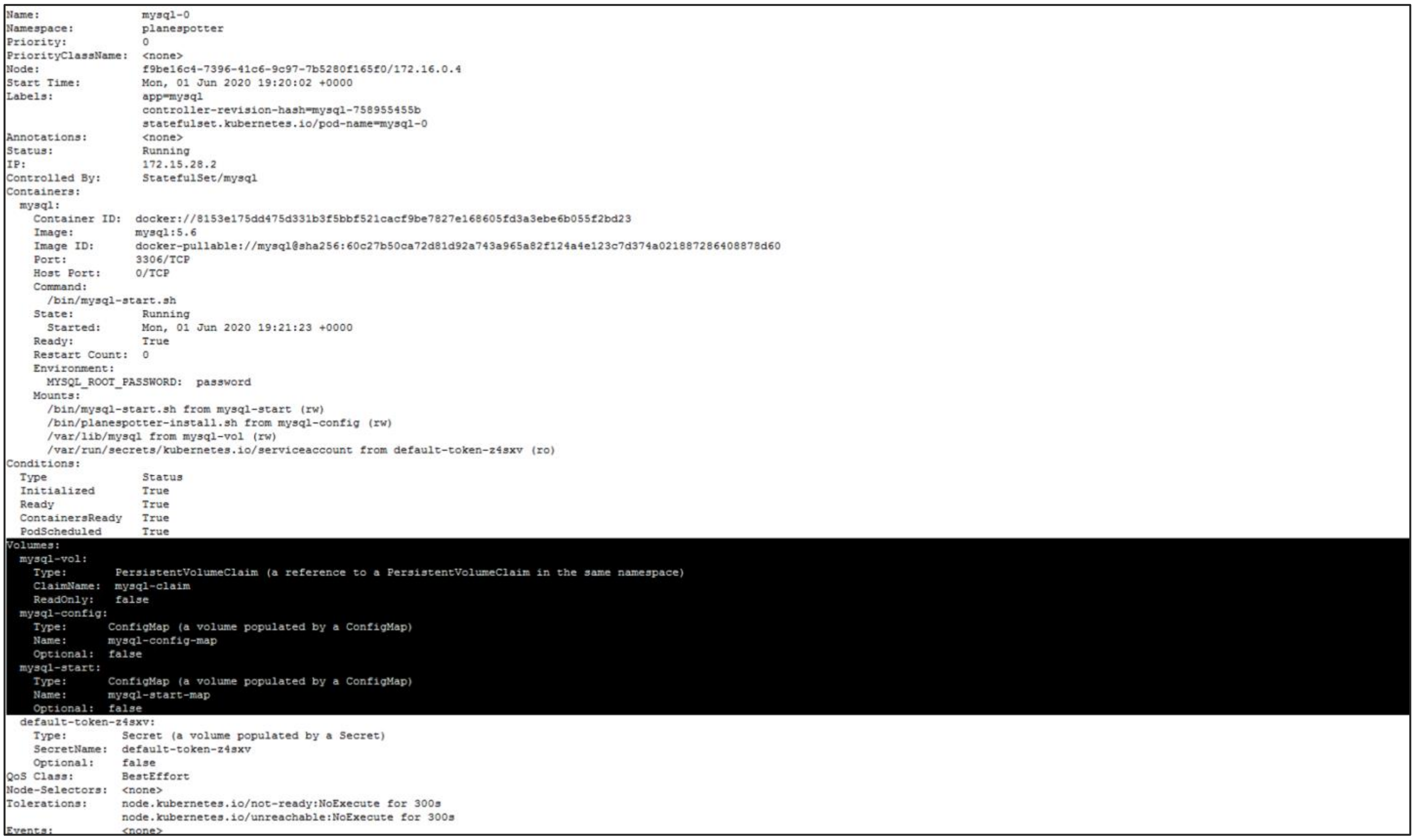
Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
ClaimName: mysql-claim
ReadOnly: false

mysql-config:

Type: ConfigMap (a volume populated by a ConfigMap)
Name: mysql-config-map
Optional: false

mysql-start:

Type: ConfigMap (a volume populated by a ConfigMap)
Name: mysql-start-map
Optional: false



```
Name:               mysql-0
Namespace:          planespotter
Priority:           0
PriorityClassName:  <none>
Node:               f9be16c4-7396-41c6-9c97-7b5280f165f0/172.16.0.4
Start Time:         Mon, 01 Jun 2020 19:20:02 +0000
Labels:             app=mysql
                    controller-revision-hash=mysql-758955455b
                    statefulset.kubernetes.io/pod-name=mysql-0
Annotations:        <none>
Status:             Running
IP:                 172.15.28.2
Controlled By:      StatefulSet/mysql
Containers:
  mysql:
    Container ID:  docker://8153e175dd475d331b3f5bbf521cacf9be7827e168605fd3a3ebe6b055f2bd23
    Image:         mysql:5.6
    Image ID:      docker-pullable://mysql@sha256:60c27b50ca72d81d92a743a965a82f124a4e123c7d374a021887286408878d60
    Port:          3306/TCP
    Host Port:     0/TCP
    Command:
      /bin/mysql-start.sh
    State:          Running
      Started:      Mon, 01 Jun 2020 19:21:23 +0000
    Ready:          True
    Restart Count:  0
    Environment:
      MYSQL_ROOT_PASSWORD:  password
    Mounts:
      /bin/mysql-start.sh from mysql-start (rw)
      /bin/planespotter-install.sh from mysql-config (rw)
      /var/lib/mysql from mysql-vol (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-z4sxv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  mysql-vol:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  mysql-claim
    ReadOnly:   false
  mysql-config:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      mysql-config-map
    Optional:  false
  mysql-start:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      mysql-start-map
    Optional:  false
  default-token-z4sxv:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-z4sxv
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:          <none>
```

Step 7: Annotate the mysql-0 pod

kubectl -n planespotter annotate pod/<mysql pod name from 4.3 eg. mysql-0> backup.velero.io/backup-volumes=mysql-vol,mysql-config,mysql-start

e.g.

kubectl -n planespotter annotate pod/mysql-0 backup.velero.io/backup-volumes=mysql-vol,mysql-config,mysql-start

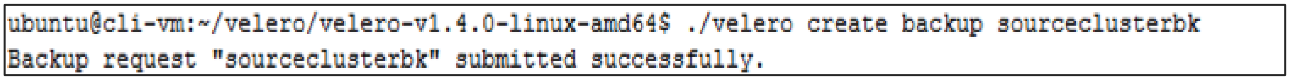
Step 8: Create a backup of the whole cluster

cd ~/velero/velero-v1.4.0-linux-amd64

./velero create backup <BACKUP NAME>

E.g.

./velero create backup sourceclusterbk



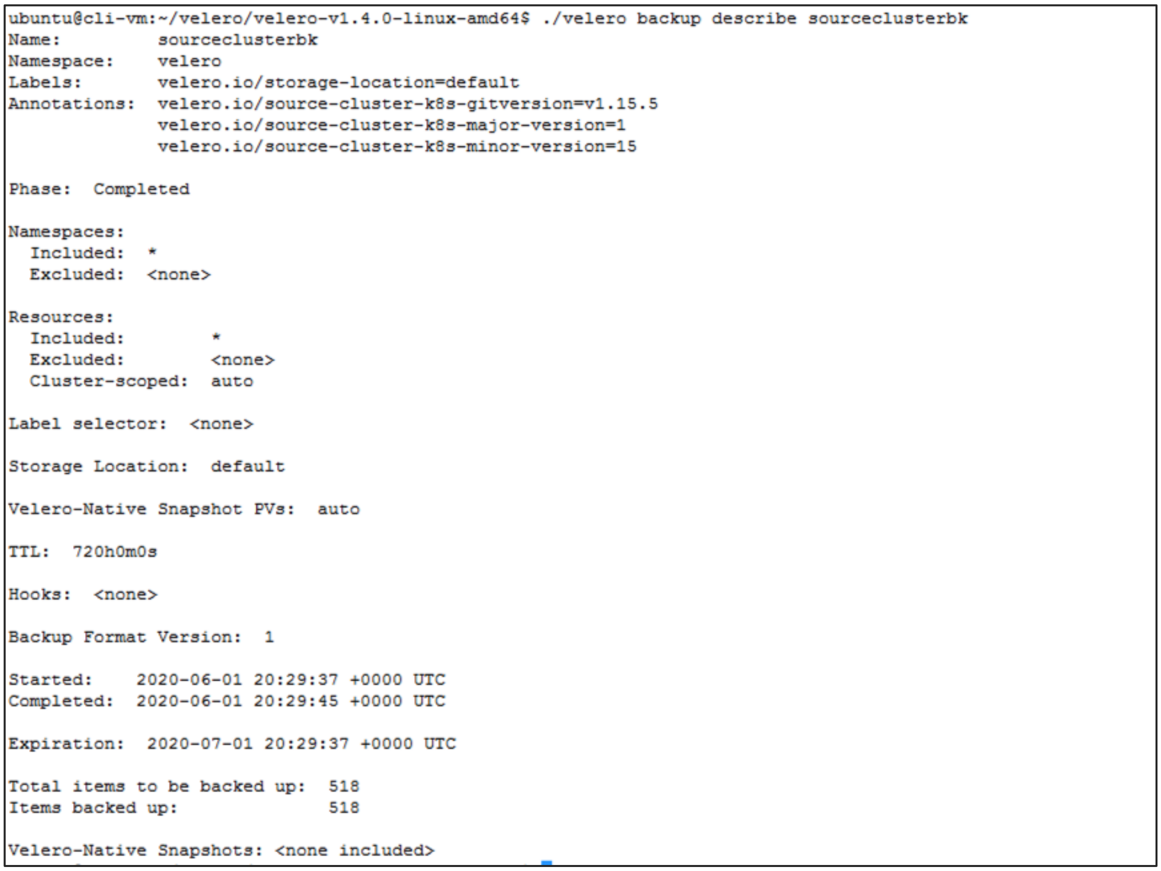
```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero create backup sourceclusterbk
Backup request "sourceclusterbk" submitted successfully.
```

Step 9: Check status of the backup

./velero backup describe <BACKUP NAME>

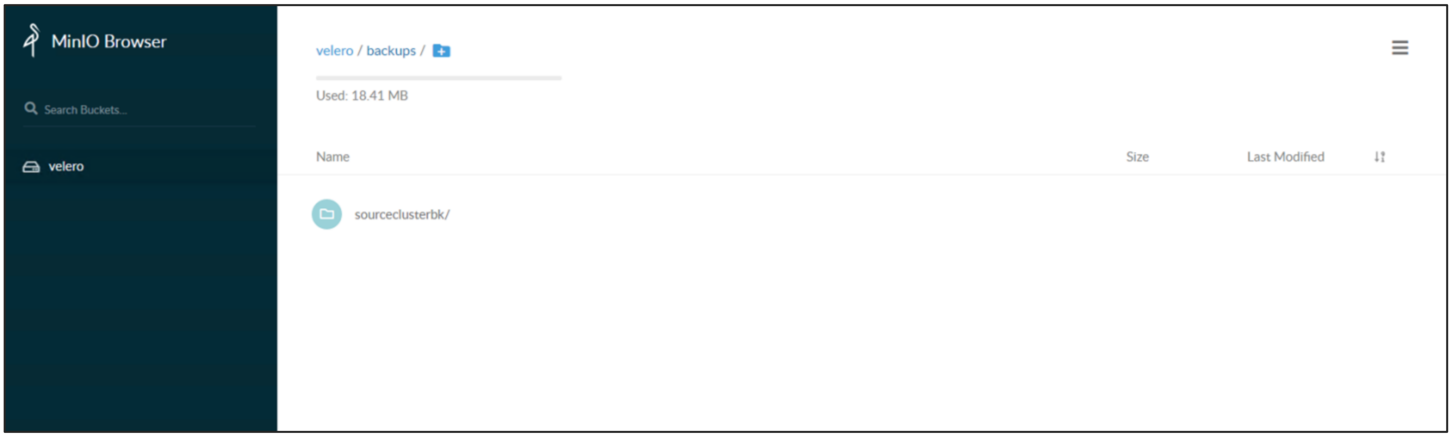
E.g.

./velero backup describe sourceclusterbk



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero backup describe sourceclusterbk
Name:         sourceclusterbk
Namespace:    velero
Labels:       velero.io/storage-location=default
Annotations:  velero.io/source-cluster-k8s-gitversion=v1.15.5
              velero.io/source-cluster-k8s-major-version=1
              velero.io/source-cluster-k8s-minor-version=15

Phase:  Completed

Namespaces:
  Included:  *
  Excluded:  <none>

Resources:
  Included:        *
  Excluded:        <none>
  Cluster-scoped:  auto

Label selector:  <none>

Storage Location:  default

Velero-Native Snapshot PVs:  auto

TTL:  720h0m0s

Hooks:  <none>

Backup Format Version:  1

Started:    2020-06-01 20:29:37 +0000 UTC
Completed:  2020-06-01 20:29:45 +0000 UTC

Expiration:  2020-07-01 20:29:37 +0000 UTC

Total items to be backed up:  518
Items backed up:              518

Velero-Native Snapshots: <none included>
```

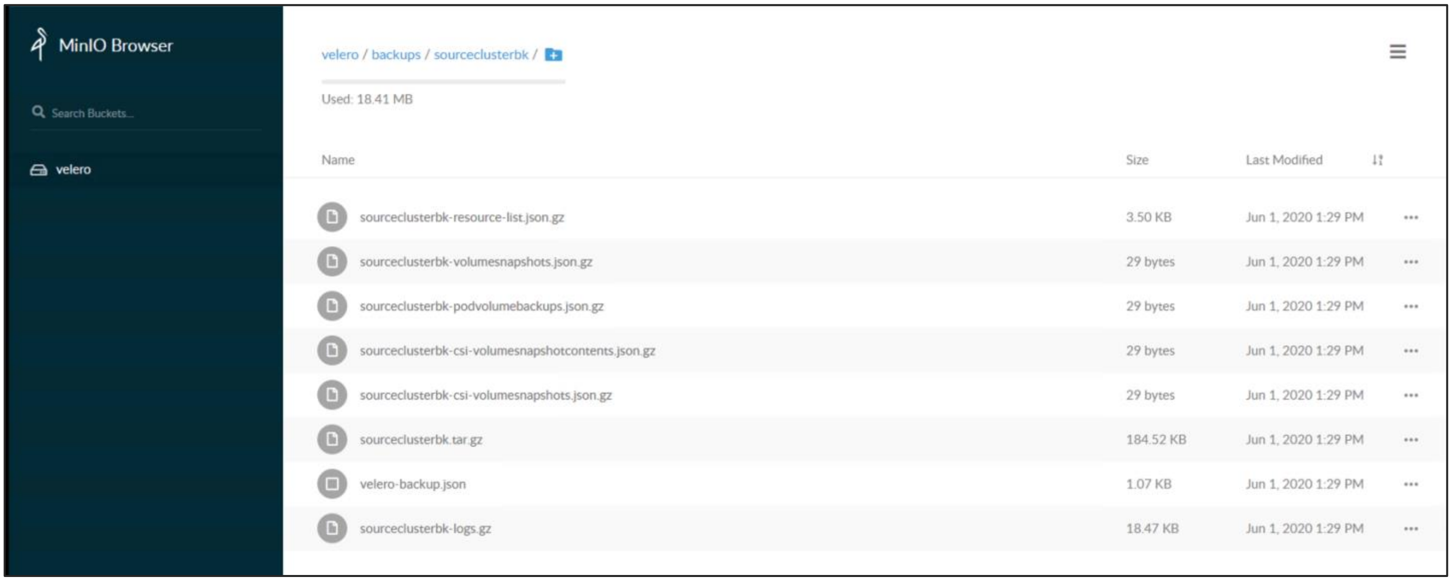
Step 10: Login to Minio and check if the backup has been created. *http://<minio-service-external-ip*> , e.g. http://10.40.14.43



velero / backups /

Used: 18.41 MB

Name	Size	Last Modified
sourceclusterbk/		



velero / backups / sourceclusterbk /

Used: 18.41 MB

Name	Size	Last Modified
sourceclusterbk-resource-list.json.gz	3.50 KB	Jun 1, 2020 1:29 PM
sourceclusterbk-volumesnapshots.json.gz	29 bytes	Jun 1, 2020 1:29 PM
sourceclusterbk-podvolumebackups.json.gz	29 bytes	Jun 1, 2020 1:29 PM
sourceclusterbk-csi-volumesnapshotcontents.json.gz	29 bytes	Jun 1, 2020 1:29 PM
sourceclusterbk-csi-volumesnapshots.json.gz	29 bytes	Jun 1, 2020 1:29 PM
sourceclusterbk.tar.gz	184.52 KB	Jun 1, 2020 1:29 PM
velero-backup.json	1.07 KB	Jun 1, 2020 1:29 PM
sourceclusterbk-logs.gz	18.47 KB	Jun 1, 2020 1:29 PM

Step 11: Create a backup of the planespotter namespace

```
cd ~/velero/velero-v1.4.0-linux-amd64
./velero backup create <BACKUP NAME> --include-namespaces <NAMESPACE1>
```

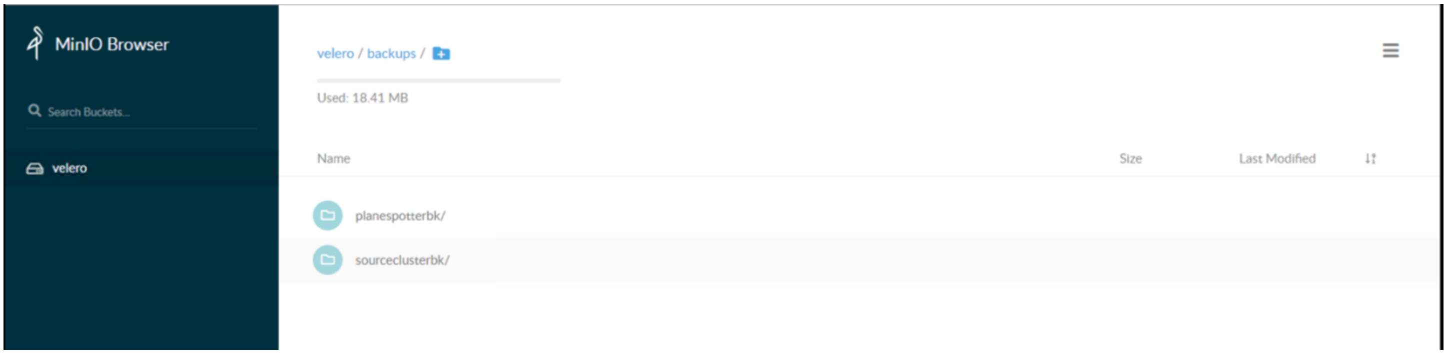
E.g.

```
./velero backup create planespotterbk --include-namespaces planespotter
```

Step 12: Check status of the backup

```
./velero backup describe planespotterbk
```

Step 13: Login to minio and check if the backup has been created.



velero / backups /

Used: 18.41 MB

Name	Size	Last Modified
planespotterbk/		
sourceclusterbk/		

Step 14: Other options for backup

```
./velero backup create planes --selector app=planespotter
```

Check Velero documentation *https://velero.io/docs/v1.4/* for other options

Restore to the Target Cluster

This section describes steps to restore a Velero backup to a target cluster . The steps give an overview of restoring a backup of a namespace and an entire clusterbackup.

Step 1: Get kube config for the source cluster

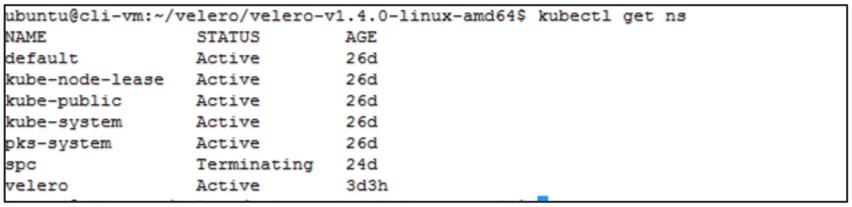
```
pks get-kubeconfig <target-cluster> -a <pks api> -u <user> -p <password> -k
E.g.
pks get-kubeconfig my-cluster -a pks.corp.local -u riaz -p VMware1! -k
```

Step 2: Set kubectl context to the target cluster

```
kubectl config use-context <target-cluster>
E.g.
kubectl config use-context my-cluster
```

Step 3: Check all resources running on the target cluster

```
kubectl get ns
```



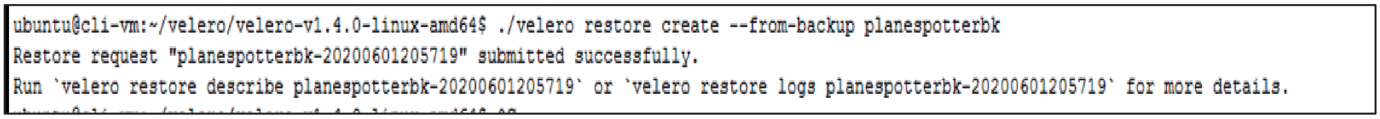
```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get ns
NAME                 STATUS        AGE
default              Active        26d
kube-node-lease      Active        26d
kube-public          Active        26d
kube-system          Active        26d
pks-system           Active        26d
spc                  Terminating   24d
velero               Active        3d3h
```

NOTE: planespotter , x1, y1 and z1 namespaces do not exist

RESTORE A NAMESPACE

Step 4: Restore the planespotter namespace from the planespotterbk created in the previous step

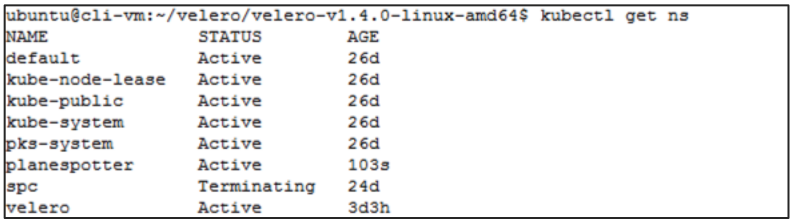
```
cd ~/velero/velero-v1.4.0-linux-amd64
./velero restore create --from-backup planespotterbk
```



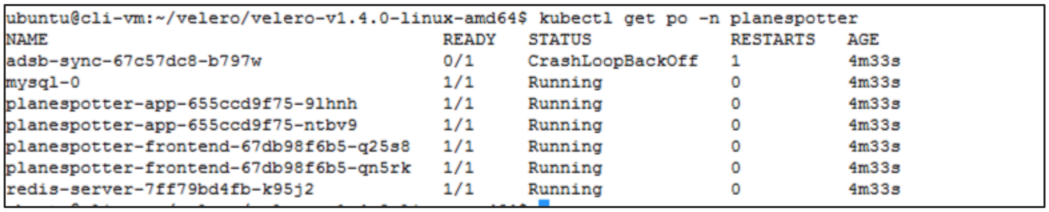
```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero restore create --from-backup planespotterbk
Restore request "planespotterbk-20200601205719" submitted successfully.
Run `velero restore describe planespotterbk-20200601205719` or `velero restore logs planespotterbk-20200601205719` for more details.
```

Step 5: Check the status of the restore in the cluster, the planespotter namespace should be created and the pods should be up and running. Make sure that the pv is also created and bound

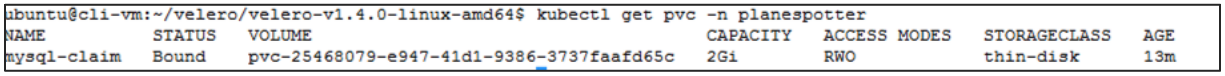
```
kubectl get ns
kubectl get po -n planespotter
kubectl get pvc -n planespotter
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get ns
NAME              STATUS        AGE
default           Active        26d
kube-node-lease   Active        26d
kube-public       Active        26d
kube-system       Active        26d
pks-system        Active        26d
planespotter      Active        103s
spc               Terminating   24d
velero            Active        3d3h
```



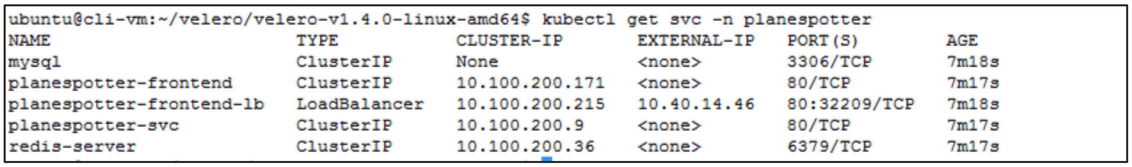
```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get po -n planespotter
NAME                                     READY   STATUS             RESTARTS   AGE
adsb-sync-67c57dc8-b797w                 0/1     CrashLoopBackOff   1          4m33s
mysql-0                                  1/1     Running            0          4m33s
planespotter-app-655ccd9f75-91hnh        1/1     Running            0          4m33s
planespotter-app-655ccd9f75-ntbv9        1/1     Running            0          4m33s
planespotter-frontend-67db98f6b5-q25s8   1/1     Running            0          4m33s
planespotter-frontend-67db98f6b5-qn5rk   1/1     Running            0          4m33s
redis-server-7ff79bd4fb-k95j2            1/1     Running            0          4m33s
```



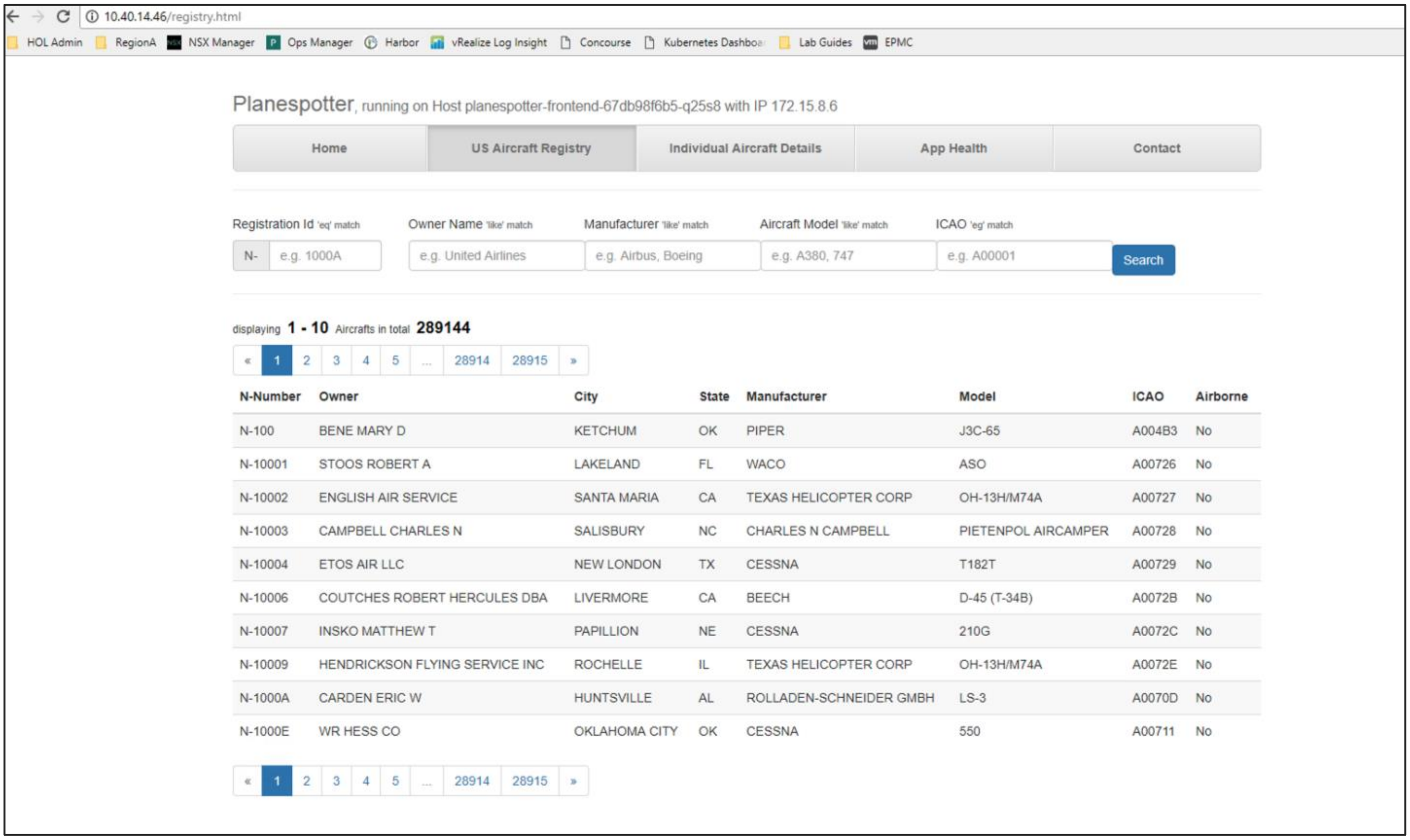
```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get pvc -n planespotter
NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mysql-claim   Bound    pvc-25468079-e947-41d1-9386-3737faafd65c   2Gi        RWO            thin-disk      13m
```

Step 6: Get the external ip for the planespotter app and point the browser to it and make sure all the data is visible in the application and the application is reachable

kubectl get svc -n planespotter



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get svc -n planespotter
NAME                       TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
mysql                      ClusterIP      None             <none>        3306/TCP       7m18s
planespotter-frontend      ClusterIP      10.100.200.171   <none>        80/TCP         7m17s
planespotter-frontend-lb   LoadBalancer   10.100.200.215   10.40.14.46   80:32209/TCP   7m18s
planespotter-svc           ClusterIP      10.100.200.9     <none>        80/TCP         7m17s
redis-server               ClusterIP      10.100.200.36    <none>        6379/TCP       7m17s
```



Planespotter, running on Host planespotter-frontend-67db98f6b5-q25s8 with IP 172.15.8.6

Registration Id 'eq' match: N- e.g. 1000A
Owner Name 'like' match: e.g. United Airlines
Manufacturer 'like' match: e.g. Airbus, Boeing
Aircraft Model 'like' match: e.g. A380, 747
ICAO 'eq' match: e.g. A00001

displaying **1 - 10** Aircrafts in total **289144**

N-Number	Owner	City	State	Manufacturer	Model	ICAO	Airborne
N-100	BENE MARY D	KETCHUM	OK	PIPER	J3C-65	A004B3	No
N-10001	STOOS ROBERT A	LAKELAND	FL	WACO	ASO	A00726	No
N-10002	ENGLISH AIR SERVICE	SANTA MARIA	CA	TEXAS HELICOPTER CORP	OH-13H/M74A	A00727	No
N-10003	CAMPBELL CHARLES N	SALISBURY	NC	CHARLES N CAMPBELL	PIETENPOL AIRCAMPER	A00728	No
N-10004	ETOS AIR LLC	NEW LONDON	TX	CESSNA	T182T	A00729	No
N-10006	COUTCHES ROBERT HERCULES DBA	LIVERMORE	CA	BEECH	D-45 (T-34B)	A0072B	No
N-10007	INSKO MATTHEW T	PAPILLION	NE	CESSNA	210G	A0072C	No
N-10009	HENDRICKSON FLYING SERVICE INC	ROCHELLE	IL	TEXAS HELICOPTER CORP	OH-13H/M74A	A0072E	No
N-1000A	CARDEN ERIC W	HUNTSVILLE	AL	ROLLADEN-SCHNEIDER GMBH	LS-3	A0070D	No
N-1000E	WR HESS CO	OKLAHOMA CITY	OK	CESSNA	550	A00711	No

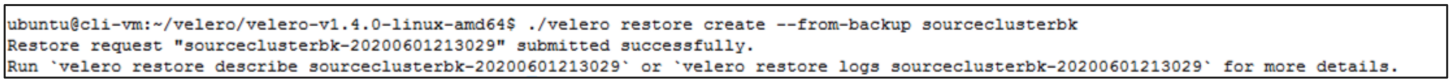
Step 7: Delete the planespotter namespace which will delete the application and the PV

```
kubectl delete ns planespotter
```

RESTORE THE CLUSTER BACKUP

Step 8: Restore the back of all resources from the source cluster to the target cluster

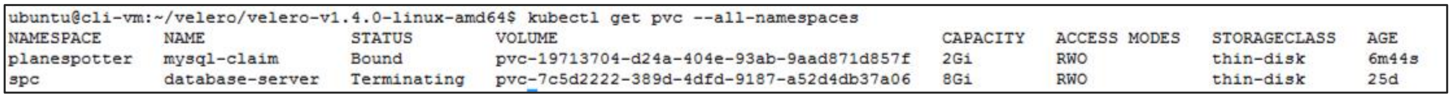
```
cd ~/velero/velero-v1.4.0-linux-amd64
./velero restore create --from-backup sourceclusterbk
```



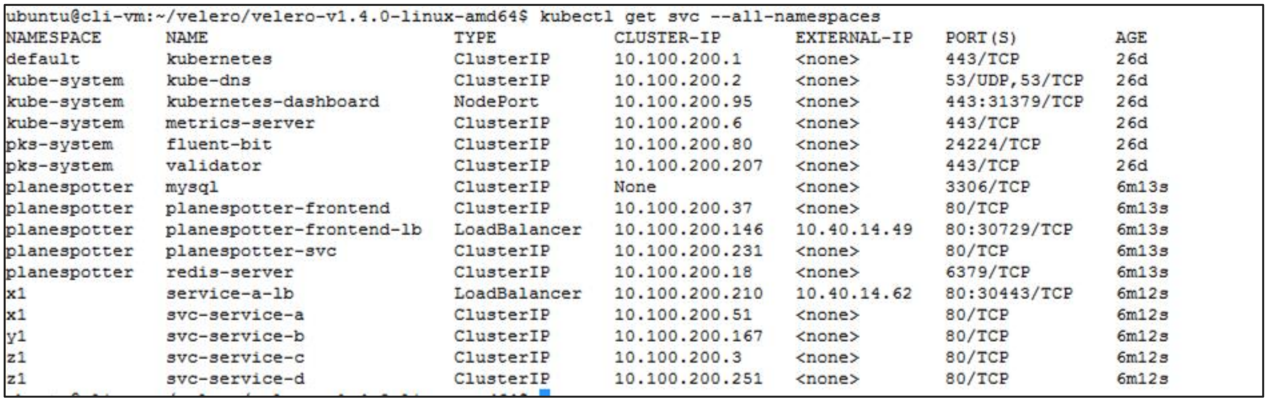
```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero restore create --from-backup sourceclusterbk
Restore request "sourceclusterbk-20200601213029" submitted successfully.
Run `velero restore describe sourceclusterbk-20200601213029` or `velero restore logs sourceclusterbk-20200601213029` for more details.
```

Step 9: Monitor the resources created in the target cluster. The planespotter , x1, y1 and z1 namespaces should be created. Pods,pv's, deployments and services should also be created.

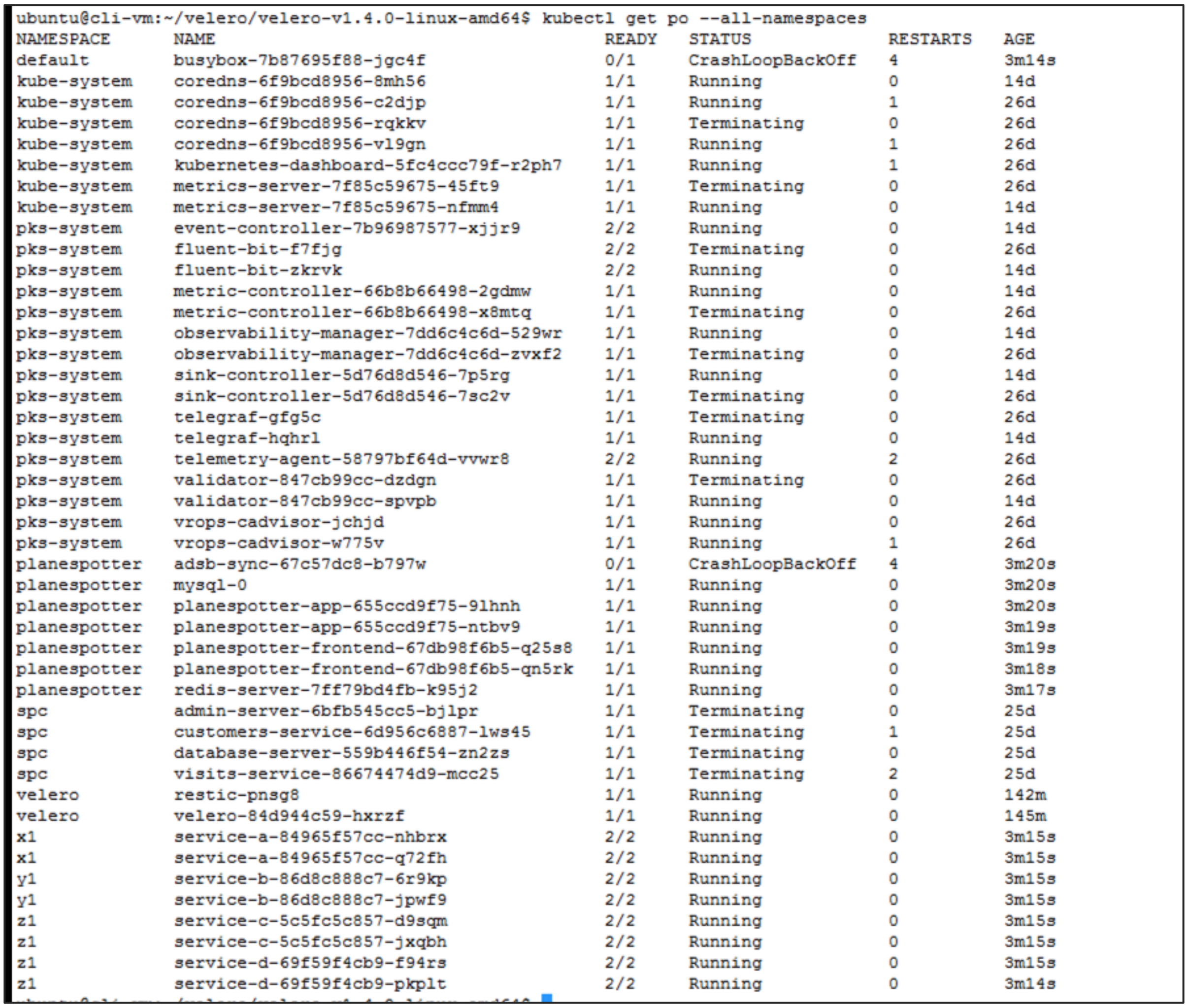
```
kubectl get ns
kubectl get po --all-namespaces
kubectl get pvc --all-namespaces
kubectl get svc --all-namespaces
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get pvc --all-namespaces
NAMESPACE      NAME              STATUS        VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
planespotter   mysql-claim       Bound         pvc-19713704-d24a-404e-93ab-9aad871d857f   2Gi        RWO            thin-disk      6m44s
spc            database-server   Terminating   pvc-7c5d2222-389d-4dfd-9187-a52d4db37a06   8Gi        RWO            thin-disk      25d
```



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get svc --all-namespaces
NAMESPACE      NAME                         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
default        kubernetes                   ClusterIP      10.100.200.1     <none>        443/TCP          26d
kube-system    kube-dns                     ClusterIP      10.100.200.2     <none>        53/UDP,53/TCP    26d
kube-system    kubernetes-dashboard         NodePort       10.100.200.95    <none>        443:31379/TCP    26d
kube-system    metrics-server               ClusterIP      10.100.200.6     <none>        443/TCP          26d
pks-system     fluent-bit                   ClusterIP      10.100.200.80    <none>        24224/TCP        26d
pks-system     validator                    ClusterIP      10.100.200.207   <none>        443/TCP          26d
planespotter   mysql                        ClusterIP      None             <none>        3306/TCP         6m13s
planespotter   planespotter-frontend        ClusterIP      10.100.200.37    <none>        80/TCP           6m13s
planespotter   planespotter-frontend-lb     LoadBalancer   10.100.200.146   10.40.14.49   80:30729/TCP     6m13s
planespotter   planespotter-svc             ClusterIP      10.100.200.231   <none>        80/TCP           6m13s
planespotter   redis-server                 ClusterIP      10.100.200.18    <none>        6379/TCP         6m13s
x1             service-a-lb                 LoadBalancer   10.100.200.210   10.40.14.62   80:30443/TCP     6m12s
x1             svc-service-a                ClusterIP      10.100.200.51    <none>        80/TCP           6m12s
y1             svc-service-b                ClusterIP      10.100.200.167   <none>        80/TCP           6m12s
z1             svc-service-c                ClusterIP      10.100.200.3     <none>        80/TCP           6m12s
z1             svc-service-d                ClusterIP      10.100.200.251   <none>        80/TCP           6m12s
```



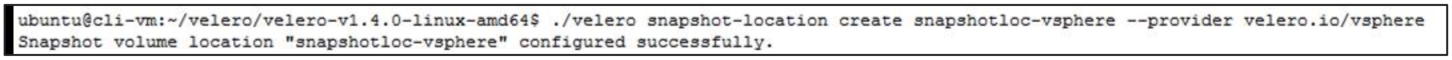
```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ kubectl get po --all-namespaces
NAMESPACE      NAME                                        READY   STATUS             RESTARTS   AGE
default        busybox-7b87695f88-jgc4f                    0/1     CrashLoopBackOff   4          3m14s
kube-system    coredns-6f9bcd8956-8mh56                    1/1     Running            0          14d
kube-system    coredns-6f9bcd8956-c2djp                    1/1     Running            1          26d
kube-system    coredns-6f9bcd8956-rqkkv                    1/1     Terminating        0          26d
kube-system    coredns-6f9bcd8956-vl9gn                    1/1     Running            1          26d
kube-system    kubernetes-dashboard-5fc4ccc79f-r2ph7       1/1     Running            1          26d
kube-system    metrics-server-7f85c59675-45ft9             1/1     Terminating        0          26d
kube-system    metrics-server-7f85c59675-nfmm4             1/1     Running            0          14d
pks-system     event-controller-7b96987577-xjjr9           2/2     Running            0          14d
pks-system     fluent-bit-f7fjg                            2/2     Terminating        0          26d
pks-system     fluent-bit-zkrvk                            2/2     Running            0          14d
pks-system     metric-controller-66b8b66498-2gdmw          1/1     Running            0          14d
pks-system     metric-controller-66b8b66498-x8mtq          1/1     Terminating        0          26d
pks-system     observability-manager-7dd6c4c6d-529wr       1/1     Running            0          14d
pks-system     observability-manager-7dd6c4c6d-zvxf2       1/1     Terminating        0          26d
pks-system     sink-controller-5d76d8d546-7p5rg            1/1     Running            0          14d
pks-system     sink-controller-5d76d8d546-7sc2v            1/1     Terminating        0          26d
pks-system     telegraf-gfg5c                              1/1     Terminating        0          26d
pks-system     telegraf-hqhrl                              1/1     Running            0          14d
pks-system     telemetry-agent-58797bf64d-vvwr8            2/2     Running            2          26d
pks-system     validator-847cb99cc-dzdgn                   1/1     Terminating        0          26d
pks-system     validator-847cb99cc-spvpb                   1/1     Running            0          14d
pks-system     vrops-cadvisor-jchjd                        1/1     Running            0          26d
pks-system     vrops-cadvisor-w775v                        1/1     Running            1          26d
planespotter   adsb-sync-67c57dc8-b797w                    0/1     CrashLoopBackOff   4          3m20s
planespotter   mysql-0                                     1/1     Running            0          3m20s
planespotter   planespotter-app-655ccd9f75-9lhnh           1/1     Running            0          3m20s
planespotter   planespotter-app-655ccd9f75-ntbv9           1/1     Running            0          3m19s
planespotter   planespotter-frontend-67db98f6b5-q25s8      1/1     Running            0          3m19s
planespotter   planespotter-frontend-67db98f6b5-qn5rk      1/1     Running            0          3m18s
planespotter   redis-server-7ff79bd4fb-k95j2               1/1     Running            0          3m17s
spc            admin-server-6bfb545cc5-bjlpr               1/1     Terminating        0          25d
spc            customers-service-6d956c6887-lws45          1/1     Terminating        1          25d
spc            database-server-559b446f54-zn2zs            1/1     Terminating        0          25d
spc            visits-service-86674474d9-mcc25             1/1     Terminating        2          25d
velero         restic-pnsg8                                1/1     Running            0          142m
velero         velero-84d944c59-hxrzf                      1/1     Running            0          145m
x1             service-a-84965f57cc-nhbrx                  2/2     Running            0          3m15s
x1             service-a-84965f57cc-q72fh                  2/2     Running            0          3m15s
y1             service-b-86d8c888c7-6r9kp                  2/2     Running            0          3m15s
y1             service-b-86d8c888c7-jpwf9                  2/2     Running            0          3m15s
z1             service-c-5c5fc5c857-d9sqm                  2/2     Running            0          3m15s
z1             service-c-5c5fc5c857-jxqbh                  2/2     Running            0          3m15s
z1             service-d-69f59f4cb9-f94rs                  2/2     Running            0          3m15s
z1             service-d-69f59f4cb9-pkplt                  2/2     Running            0          3m14s
```

Step 10: Get the external ip for the planespotter app and point the browser to it and make sure all the data is visible in the application and the application is reachable

```
kubectl get svc -n planespotter
```

Snapshots

Snapshots in a vsphere environment are created using the vsphere plugin for velero. This plugin is a volume snapshotter plugin that provides crash-consistent snapshots of vSphere block volumes and backup of volume data into S3 compatible storage (Minio)



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero snapshot-location create snapshotloc-vsphere --provider velero.io/vsphere
Snapshot volume location "snapshotloc-vsphere" configured successfully.
```

Compatibility

- Velero - Version 1.3.2 or above
- vSphere - Version 6.7U3 or above
- vSphere CSI/CNS driver 1.0.2 or above
- Kubernetes 1.14 or above (note: the Velero Plug-in for vSphere does not support Guest or Supervisor clusters on vSphere yet)

Step 1: Add the vSphere velero plugin

```
cd ~/velero/velero-v1.4.0-linux-amd64

./velero plugin add vsphereveleroplugin/velero-plugin-for-vsphere:1.0.0
```

Step 2: Create a volume Snapshot location

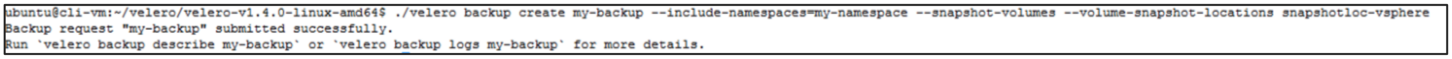
./velero snapshot-location create <snapshot location name> --provider velero.io/vsphere

e.g.

./velero snapshot-location create **snapshotloc-vsphere** --provider velero.io/vsphere

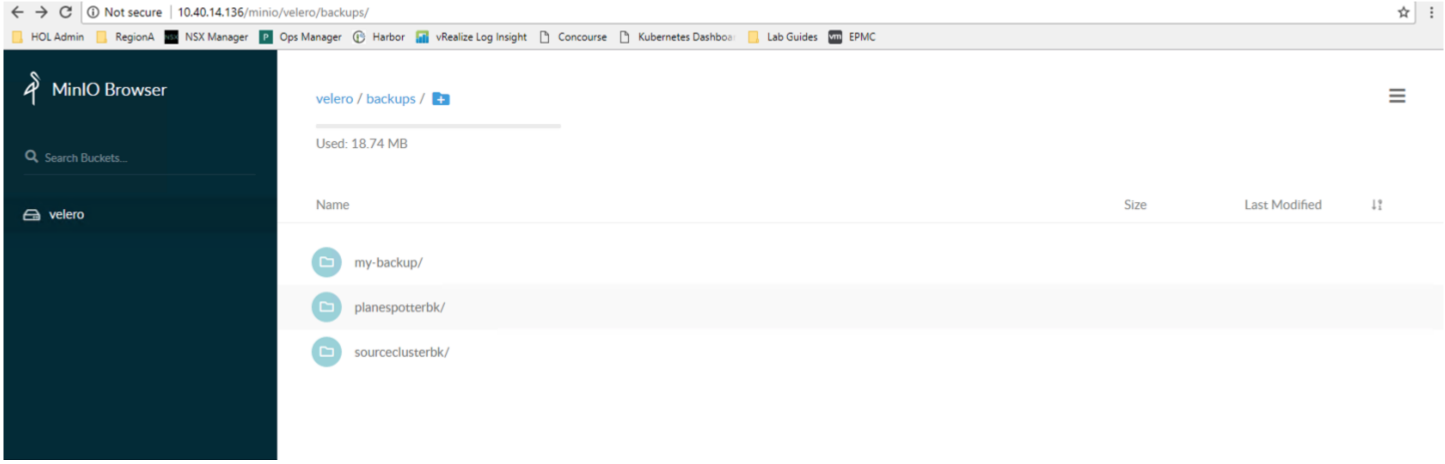
Step 3: Run a Velero backup and specify the --snapshot-volumes flag and specify the VolumeSnapshotLocation. Use the snapshot location created above

./velero backup create my-backup --include-namespaces=my-namespace --snapshot-volumes --volume-snapshot-locations **snapshotloc-vsphere**



```
ubuntu@cli-vm:~/velero/velero-v1.4.0-linux-amd64$ ./velero backup create my-backup --include-namespaces=my-namespace --snapshot-volumes --volume-snapshot-locations snapshotloc-vsphere
Backup request "my-backup" submitted successfully.
Run `velero backup describe my-backup` or `velero backup logs my-backup` for more details.
```

Step 4: Login to Minio and check the id the backup was created



my-backup/
planespotterbk/
sourceclusterbk/

Backup will complete after the local snapshots have completed, and your Kubernetes metadata has been uploaded to the object store specified. At this point, all of the data may not have been uploaded to your S3 object store. Data movement happens in the background and may take a significant amount of time to complete.

Step 5: Restore follows the same steps as above

./velero restore create --from-backup my-backup

More info on Snapshots can be found on *https://github.com/vmware-tanzu/velero-plugin-for-vsphere*

Conclusion

We hope this document was useful. As you try these configuration steps, please provide any feedback or questions in the comments section for this document on code.vmware.com. Also, do let us know if you have any suggestions or if you would like to see guidance on other topics.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.