Kubernetes Configs: ConfigMaps, ETCD backups, Velero HA EKS backup&restore with terraform, Secrets

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Ask Personal AI Document assistant to learn interactively (FASTER)!

1 Download PDF

2 Go to website

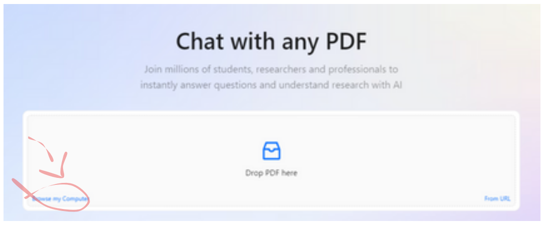
3 Browse file

4 Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesConfigs.pdf

Click there to go to ChatPdf website 2

3



Chat with any PDF

Join millions of students, researchers and professionals to instantly answer questions and understand research with AI

Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes Configs: ConfigMaps

1) Intro

ConfigMaps are read-only volumes that store structured data, making them ideal for injecting configuration into containers. Unlike other persistent volumes, they can't be written to but can be transformed into environment variables or other formats for flexible use.

check example:

```
cat volume/config-map.yaml


cd kubernetes-demo
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: silly-demo
  labels:
    app.kubernetes.io/name: silly-demo
data:
  videos.yaml: |
    - id: "1"
      title: something
    - id: "2"
      title: else
    - id: "3"
      title: something new
  message: Hello, DevOps Toolkit!
```

We define a ConfigMap silly-demo with two data entries: videos.yaml, containing YAML text, and message, containing plain text. All ConfigMap values are always strings.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: silly-demo
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        - image: ghcr.io/vfarcic/silly-demo:1.4.235-alpine
          ...
          envFrom:
            - configMapRef:
                name: silly-demo
          env:
            - name: MESSAGE
              valueFrom:
                configMapKeyRef:
                  name: silly-demo
                  key: message
          volumeMounts:
            - name: cache
              mountPath: /cache
      volumes:
        - name: cache
          configMap:
            name: silly-demo
```

In the Deployment, we define a configMap volume and mount it like other volumes, but with the added ability to extract its data.

We specify that all ConfigMap values should be converted into environment variables using envFrom.

A ConfigMap key can also be mapped to a custom environment variable name, like setting MESSAGE from the silly-demo ConfigMap's message key.

This setup provides files from ConfigMap keys, converts all keys to environment variables, and maps message to MESSAGE.

Apply manifest

```
kubectl apply \
   --filename volume/config-map.yaml
```

check configmaps

```
kubectl --namespace a-team get configmaps
```

The ConfigMap was successfully created as expected.

To view the container's environment variables, run env inside it.

```
kubectl --namespace a-team exec service/silly-demo \
   --stdin --tty -- env
```

The message and videos.yaml variables are present, created via envFrom to convert all ConfigMap entries into environment variables.

The MESSAGE variable is also there, explicitly mapped to the message key in the ConfigMap.

Additionally, files should exist in the /cache directory. Let's list them inside the container.

```
kubectl --namespace a-team exec service/silly-demo \
   --stdin --tty -- ls /cache/
```

The message and videos.yaml files are present, as we mounted the ConfigMap, converting each key into a file.

To confirm, let's check the content of the videos.yaml file.

```
kubectl --namespace a-team exec service/silly-demo \
    --stdin --tty -- cat /cache/videos.yaml \
    | yq .
```

The file content matches the key value in the ConfigMap. Remove the ConfigMap.

```
kubectl --namespace a-team delete \
    --filename volume/config-map.yaml
```

Kubernetes Configs: ETCD backup & restore

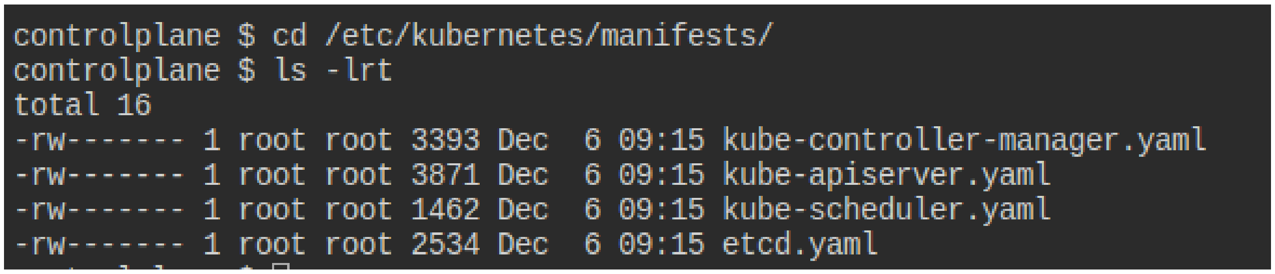
1) Intro

ETCD backup and restore involve creating snapshots of the ETCD key-value store, which holds critical data for Kubernetes clusters, such as cluster state, configurations, secrets, and metadata. Regular backups of ETCD ensure that if the cluster encounters failure, corruption, or data loss, you can restore the entire cluster state to a previous point.

2) How to take backup

Navigate to /etc/kubernetes/manifests, a critical directory on a Kubernetes node that contains static pod definitions for the cluster.

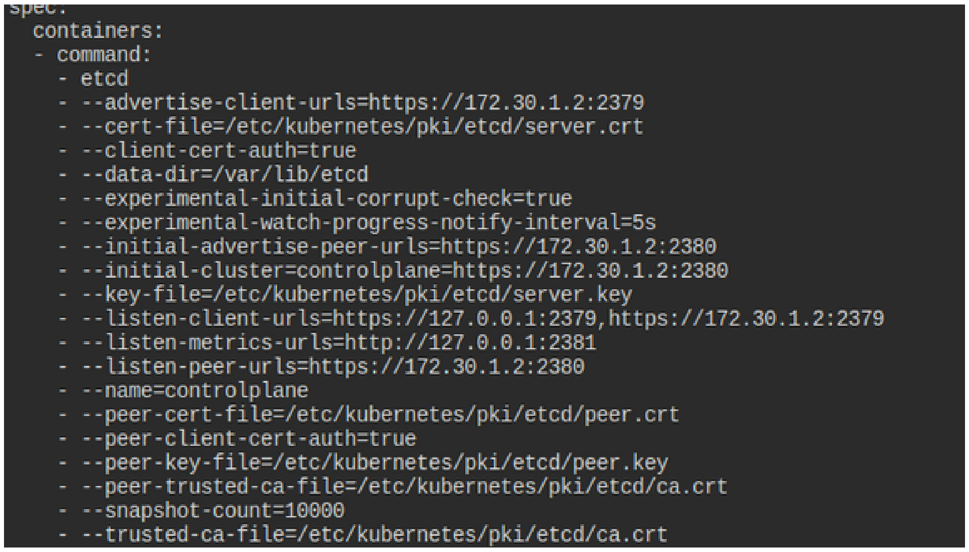
```
cd /etc/kubernetes/manifests/

ls -lrt
```



```
controlplane $ cd /etc/kubernetes/manifests/
controlplane $ ls -lrt
total 16
-rw------- 1 root root 3393 Dec  6 09:15 kube-controller-manager.yaml
-rw------- 1 root root 3871 Dec  6 09:15 kube-apiserver.yaml
-rw------- 1 root root 1462 Dec  6 09:15 kube-scheduler.yaml
-rw------- 1 root root 2534 Dec  6 09:15 etcd.yaml
```

show etcd.yaml

```
sudo less etcd.yaml
```



```
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://172.30.1.2:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --client-cert-auth=true
    - --data-dir=/var/lib/etcd
    - --experimental-initial-corrupt-check=true
    - --experimental-watch-progress-notify-interval=5s
    - --initial-advertise-peer-urls=https://172.30.1.2:2380
    - --initial-cluster=controlplane=https://172.30.1.2:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --listen-client-urls=https://127.0.0.1:2379,https://172.30.1.2:2379
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://172.30.1.2:2380
    - --name=controlplane
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
```

Most important parameters need for ETCD backup:

--advertise-client-urls=https://172.30.1.2:2379: Specifies the URL where the etcd server advertises its client endpoints for access by external clients.

--data-dir=/var/lib/etcd: Defines the directory where etcd stores its data, including the snapshot and WAL (Write-Ahead Log).

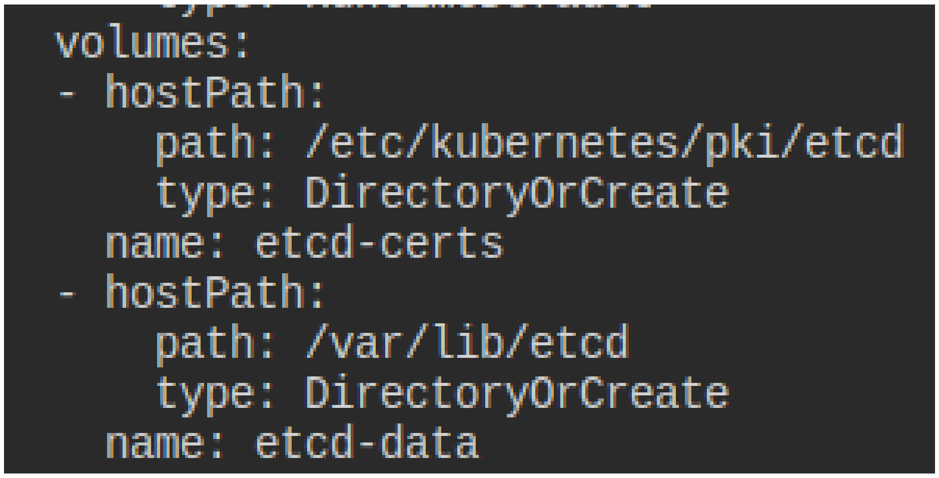
--listen-client-urls=https://127.0.0.1:2379,https://172.30.1.2:2379: Specifies the URLs where the etcd server listens for client requests, both local and external.

--key-file=/etc/kubernetes/pki/etcd/server.key: Points to the private key file used by etcd for securing client-server communication over HTTPS.

--cert-file=/etc/kubernetes/pki/etcd/server.crt: Specifies the certificate file that authenticates the etcd server to clients.

--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt: Indicates the CA certificate file used to verify the authenticity of client certificates during mutual TLS authentication.

All important data is stored on host in volumes /etc/kubernetes/pki/etcd and /var/lib/etcd.



```
  volumes:
  - hostPath:
      path: /etc/kubernetes/pki/etcd
      type: DirectoryOrCreate
    name: etcd-certs
  - hostPath:
      path: /var/lib/etcd
      type: DirectoryOrCreate
    name: etcd-data
```

Backups are performed using the etcdctl tool, which provides commands to snapshot the etcd database and restore it when needed.

```
sudo apt install etcd-client
```

To set the proper etcdctl API version, use the environment variable ETCDCTL_API before running any etcdctl commands

```
export ETCDCTL_API=3
```

To use etcdctl for taking a backup, you need to specify important parameters like endpoints, certificates, and the directory to store the backup

```
sudo etcdctl --endpoint=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
snapshot save /opt/etcd-backup.db
```

Check snapshot weight

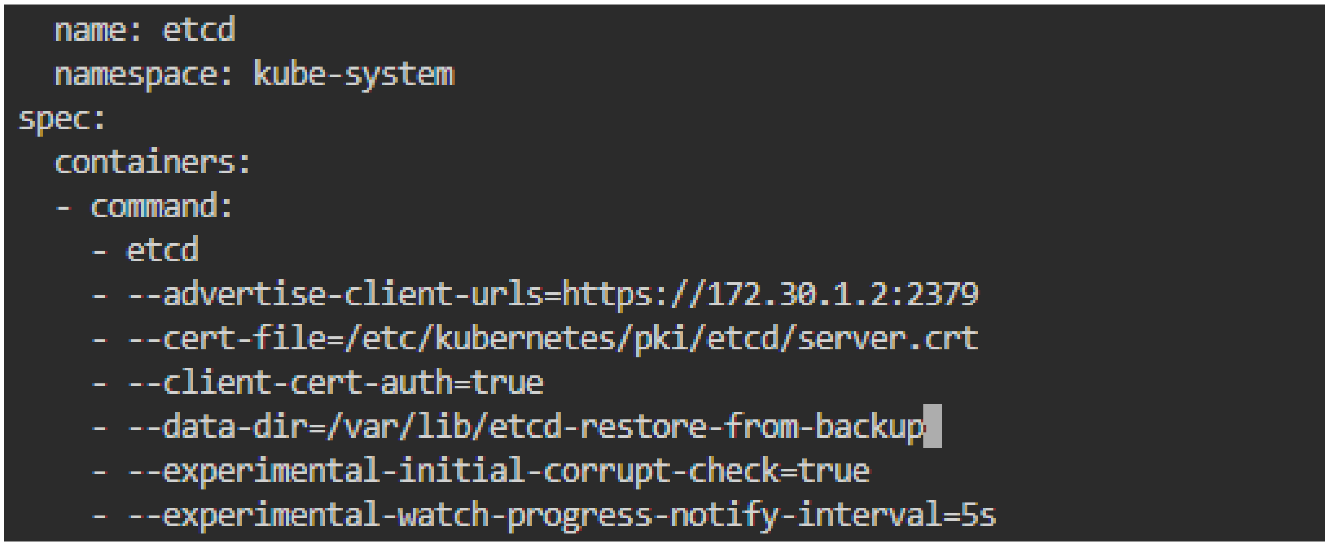
```
du -sh /opt/etcd-backup.db
```

restore to new etcd dir:

```
sudo etcdctl --endpoint=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
snapshot restore /opt/etcd-backup.db --data-dir=/var/lib/etcd-restore-from-backup
```

change etcd.yaml to point for new etcd directory and configure volumes:

```
cd /etc/kubernetes/manifests
nano etcd.yaml
```



```
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://172.30.1.2:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --client-cert-auth=true
    - --data-dir=/var/lib/etcd-restore-from-backup
    - --experimental-initial-corrupt-check=true
    - --experimental-watch-progress-notify-interval=5s
```

restart api server and kubectl so changes will be applied:

```
cd /etc/kubernetes/manifest/
sudo mv * /tmp
sudo mv /tmp/*.yaml .
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

All data should be succesfully restored. Check if pods, deployment or any other workloads you backed up, functions properly.

Kubernetes Configs: Velero backup & restore on EKS with terraform

1) Intro and infra & Velero setup

In cloud Kubernetes solutions, direct ETCD management isn't possible. Instead, you must use Velero for backup and restore. Velero is designed for cloud environments, providing backup solutions for both Kubernetes resources and persistent volumes. It integrates with cloud storage, ensuring reliable and scalable backups without needing to manage ETCD manually.

At first install and configure important tools, as said in Karpenter demo:

Getting Started with Karpenter

Set up a cluster and add Karpenter

https://karpenter.sh/docs/

Download and install velero bianries at host

```
VERSION=v1.15.1
wget https://github.com/vmware-
tanzu/velero/releases/download/${VERSION}/velero-${VERSION}-linux-amd64.tar.gz
```

```
tar -xvf velero-${VERSION}-linux-amd64.tar.gz
sudo mv velero-${VERSION}-linux-amd64/velero /usr/local/bin/
```

Create 2 EKS clusters. You need to pass s3 bucket name in us-east-1 as backup storage. Pass existing bucket name or non-exsistent bucket name and one will be created:

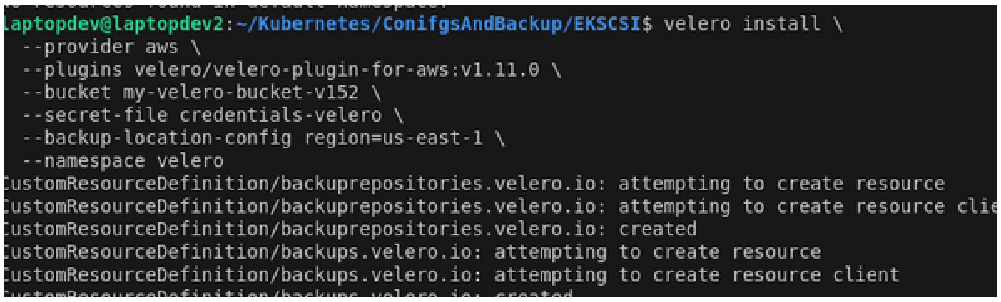
```
git clone https://github.com/MichaelRobotics/Kubernetes.git
cd Kubernetes/ConifgsAndBackup/EKSCSI
./EKSinit.sh <bucket_name>
```

Deploy velero in cluster located at us-east-2 zone. At first switch kubectl context to point at second cluster:

```
aws eks --region us-east-2 update-kubeconfig --name demo2
```

Install velero resources on k8s and specify s3 bucket location and name

```
velero install \
  --provider aws \
  --plugins velero/velero-plugin-for-aws:v1.11.0 \
  --bucket <your_bucket_name> \
  --secret-file credentials-velero \
  --backup-location-config region=us-east-1 \
  --namespace velero
```



```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI$ velero install \
  --provider aws \
  --plugins velero/velero-plugin-for-aws:v1.11.0 \
  --bucket my-velero-bucket-v152 \
  --secret-file credentials-velero \
  --backup-location-config region=us-east-1 \
  --namespace velero
CustomResourceDefinition/backuprepositories.velero.io: attempting to create resource
CustomResourceDefinition/backuprepositories.velero.io: attempting to create resource clie
CustomResourceDefinition/backuprepositories.velero.io: created
CustomResourceDefinition/backups.velero.io: attempting to create resource
CustomResourceDefinition/backups.velero.io: attempting to create resource client
```

Deploy velero in cluster located at us-east-1 zone. At first switch kubectl context to point at second cluster:

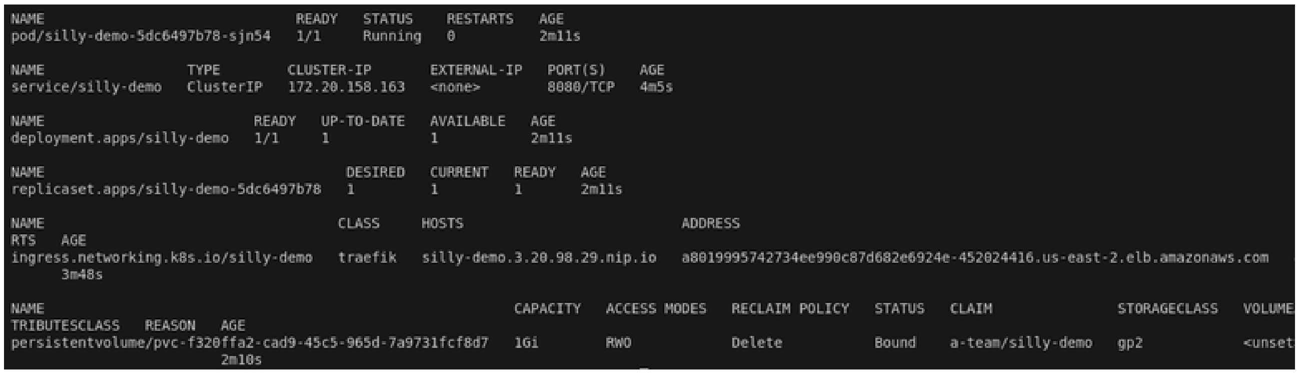
```
aws eks --region us-east-1 update-kubeconfig --name demo
```

Deploy application with persistent storage. Remember to create all resources in a-team namespace what is not mentioned in PDF.

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.
https://github.com/MichaelRobotics

Those resources should run on you k8s cluster:

```
kubectl --namespace a-team get all,ingresses,persistentvolumes
```



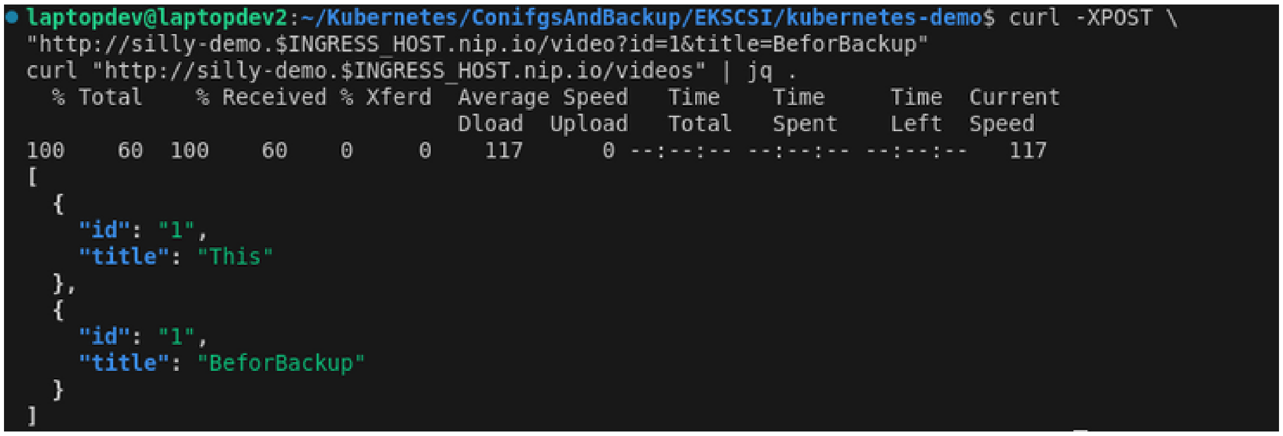
```
NAME                            READY   STATUS    RESTARTS   AGE
pod/silly-demo-5dc6497b78-sjn54   1/1     Running   0          2m11s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/silly-demo   ClusterIP   172.20.158.163   <none>        8080/TCP   4m5s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/silly-demo   1/1     1            1           2m11s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/silly-demo-5dc6497b78   1         1         1       2m11s

NAME                                     CLASS     HOSTS                         ADDRESS
RTS   AGE
ingress.networking.k8s.io/silly-demo   traefik   silly-demo.3.20.98.29.nip.io   a8019995742734ee990c87d682e6924e-452024416.us-east-2.elb.amazonaws.com
      3m48s

NAME                                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS   VOLUME
TRIBUTESCLASS   REASON   AGE
persistentvolume/pvc-f320ffa2-cad9-45c5-965d-7a9731fcf8d7   1Gi        RWO            Delete           Bound    a-team/silly-demo     gp2            <unset
                        2m10s
```

Kubernetes resources like Pods, Services, and PersistentVolumes (external storage) are essential for stateful applications that read/write data to disk. To ensure proper recovery, we need backups of both Kubernetes objects (e.g., Deployments, Pods, Secrets) and snapshots of PersistentVolume data. Without this, volumes might be restored as objects but lack data.

Logs, however, should be shipped to a dedicated store like Loki. Backing up logs within Kubernetes objects is unnecessary—focus on backing up the log storage system instead.

Push some data to external storage, so late we can test if it was actually restored:

```
curl -XPOST \
"http://silly-demo.$INGRESS_HOST.nip.io/video?id=1&title=BeforBackup"
curl "http://silly-demo.$INGRESS_HOST.nip.io/videos" | jq .
```



```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/kubernetes-demo$ curl -XPOST \
"http://silly-demo.$INGRESS_HOST.nip.io/video?id=1&title=BeforBackup"
curl "http://silly-demo.$INGRESS_HOST.nip.io/videos" | jq .
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    60  100    60    0     0    117      0 --:--:-- --:--:-- --:--:--   117
[
  {
    "id": "1",
    "title": "This"
  },
  {
    "id": "1",
    "title": "BeforBackup"
  }
]
```

Install velero on cluster:

```
velero install \
 --provider aws \
 --plugins velero/velero-plugin-for-aws:v1.11.0 \
 --bucket <your_bucket_name> \
 --secret-file credentials-velero \
 --backup-location-config region=us-east-1 \
 --namespace velero
```

2) Create backup

Backups can be scheduled or on-demand, and both are commonly used. With Velero, backups can be managed via its CLI or by applying Kubernetes resources.

To schedule a backup using the CLI, run velero schedule create, specify a name (e.g., daily), and use the --schedule flag with a cron-style format like @daily.

```
velero schedule create daily --schedule "@daily"
```

Instead of creating Kubernetes resources with commands, define them as YAML or Helm templates. This allows version control, reviews, GitOps, and other best practices.

example:

```
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: every-5-min
spec:
  schedule: '*/5 * * * *'
  skipImmediately: false
  template:
    excludedNamespaces:
    - default
    - kube-node-lease
    - kube-public
    - kube-system
```

This schedule runs every five minutes (*/5 * * * *) and excludes default, kube-node-lease, kube-public, and kube-system namespaces. Excluding some namespaces avoids unnecessary restoration, but it's simpler to back up everything and filter later. The only downside is higher storage costs if space is a concern.

download repo with velero manifests:

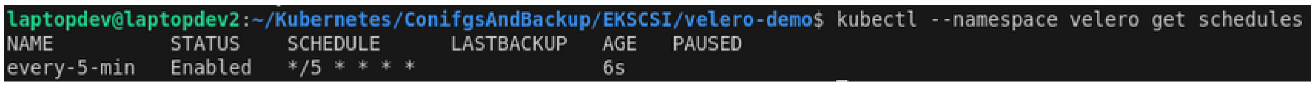
```
git clone https://github.com/vfarcic/velero-demo
cd velero-demo
```

apply manifest

```
kubectl --namespace velero apply --filename velero/schedule.yaml
```

wait for a few moments

```
kubectl --namespace velero get schedules
```



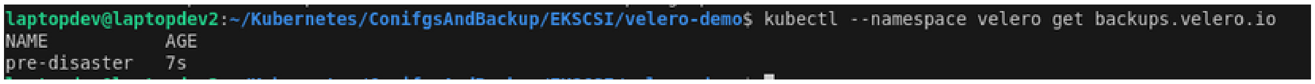
```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ kubectl --namespace velero get schedules
NAME          STATUS    SCHEDULE      LASTBACKUP   AGE   PAUSED
every-5-min   Enabled   */5 * * * *                6s
```

We created two backup schedules: one @daily and another every five minutes (*/5 * * * *), mainly to see it in action quickly. On-demand backups are also possible and useful before major changes or when anticipating issues. These can be created with the command velero backup create.

```
velero backup create pre-disaster
```

wait a moment:

```
kubectl --namespace velero get backups.velero.io
```

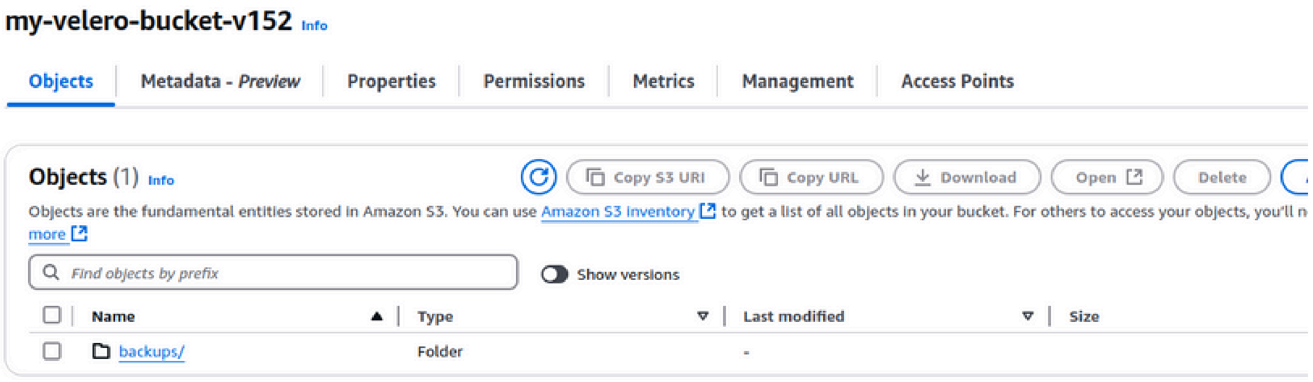


```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ kubectl --namespace velero get backups.velero.io
NAME           AGE
pre-disaster   7s
```

Here is what happened:

The Velero BackupController reacts to Backup resources created by us or a schedule. It creates snapshots of etcd, volume data, logs, and more, which are then uploaded to external storage like AWS S3, Google Cloud Storage, or others. Velero supports major cloud storage providers and S3-compatible options like Minio. You can confirm this by checking the storage bucket used for backups.

Go to AWS and check if in us-east-1 zone bucket with /backup folder exists:

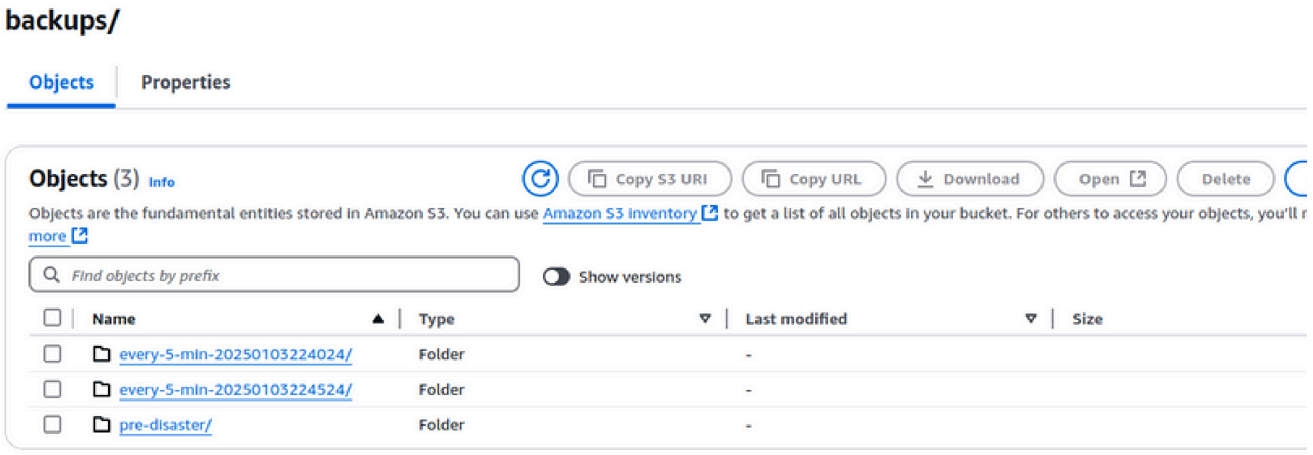

my-velero-bucket-v152 Info

Objects (1) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll n
more

Name	Type	Last modified	Size
backups/	Folder	-	

The backups directory was created, meaning we could potentially share this storage with other processes. Inside backup:


backups/

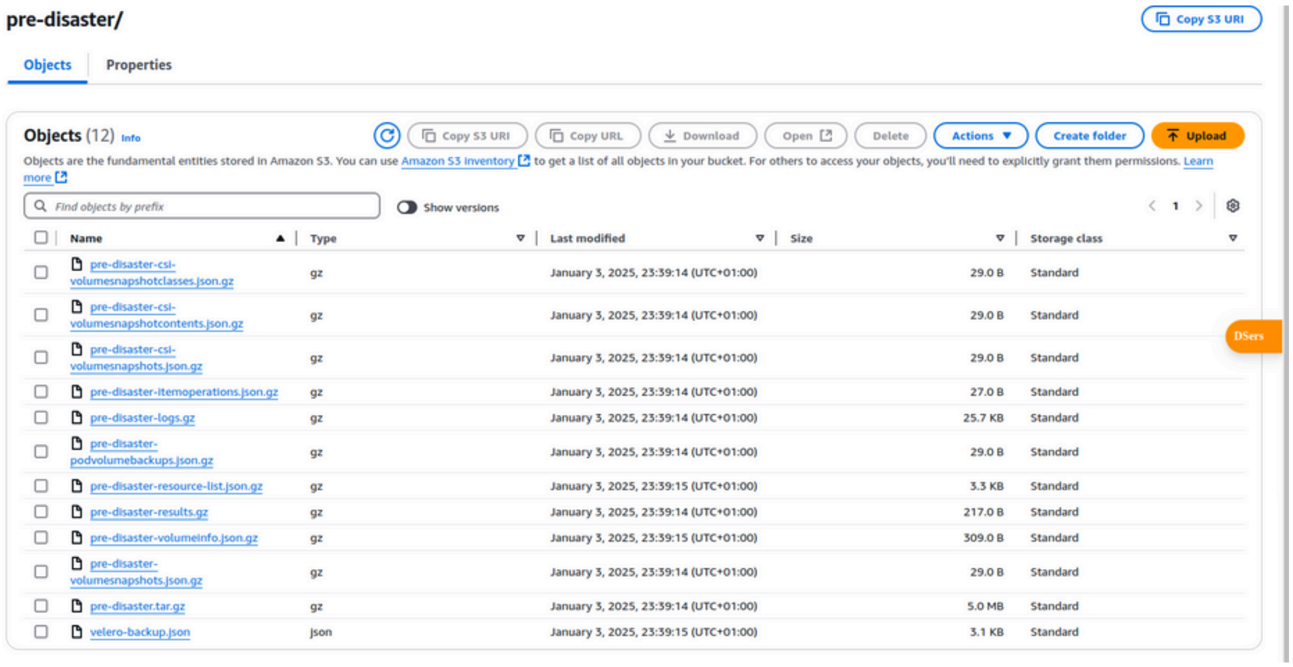
Objects (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll r
more

Name	Type	Last modified	Size
every-5-min-20250103224024/	Folder	-	
every-5-min-20250103224524/	Folder	-	
pre-disaster/	Folder	-	

there is subdir for each backup.

If we enter inside pre-disaster we can see individual files, one for each type of the resources of data that were backed up. There are volume snapshot classes (pre-disaster-csi-volumesnapshotclasses.json.gz), contents (pre-disaster-csi-volumesnapshotcontents.json.gz), and snapshots themselves (pre-disaster-csi-volumesnapshots.json.gz), item operations (pre-disaster-itemoperations.json.gz), logs (pre-disaster-logs.gz), and so on and so forth.

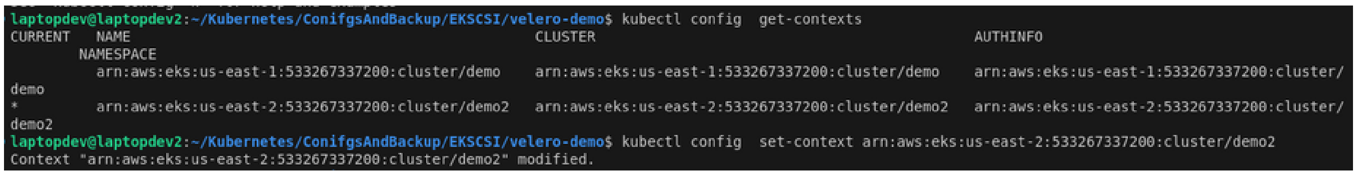


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
pre-disaster-csi-volumesnapshotclasses.json.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	29.0 B	Standard
pre-disaster-csi-volumesnapshotcontents.json.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	29.0 B	Standard
pre-disaster-csi-volumesnapshots.json.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	29.0 B	Standard
pre-disaster-itemoperations.json.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	27.0 B	Standard
pre-disaster-logs.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	25.7 KB	Standard
pre-disaster-podvolumebackups.json.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	29.0 B	Standard
pre-disaster-resource-list.json.gz	gz	January 3, 2025, 23:39:15 (UTC+01:00)	3.3 KB	Standard
pre-disaster-results.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	217.0 B	Standard
pre-disaster-volumeinfo.json.gz	gz	January 3, 2025, 23:39:15 (UTC+01:00)	309.0 B	Standard
pre-disaster-volumesnapshots.json.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	29.0 B	Standard
pre-disaster.tar.gz	gz	January 3, 2025, 23:39:14 (UTC+01:00)	5.0 MB	Standard
velero-backup.json	json	January 3, 2025, 23:39:15 (UTC+01:00)	3.1 KB	Standard

3) Restore Backup

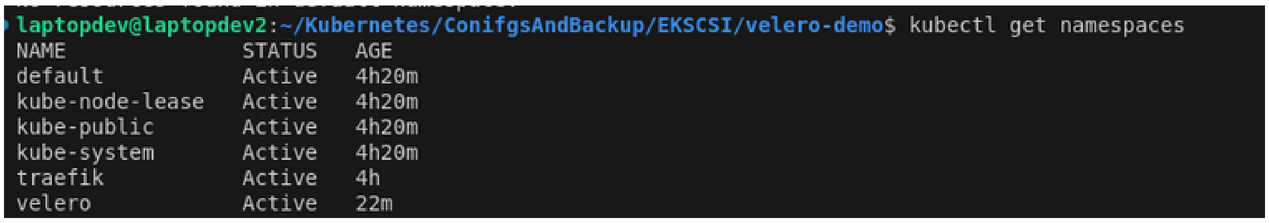
switch context to cluster on us-east-2:



```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ kubectl config  get-contexts
CURRENT   NAME                                                    CLUSTER                                                 AUTHINFO
        NAMESPACE
          arn:aws:eks:us-east-1:533267337200:cluster/demo    arn:aws:eks:us-east-1:533267337200:cluster/demo    arn:aws:eks:us-east-1:533267337200:cluster/
demo
*         arn:aws:eks:us-east-2:533267337200:cluster/demo2   arn:aws:eks:us-east-2:533267337200:cluster/demo2   arn:aws:eks:us-east-2:533267337200:cluster/
demo2
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ kubectl config  set-context arn:aws:eks:us-east-2:533267337200:cluster/demo2
Context "arn:aws:eks:us-east-2:533267337200:cluster/demo2" modified.
```

nothing is there except velero and traefik:

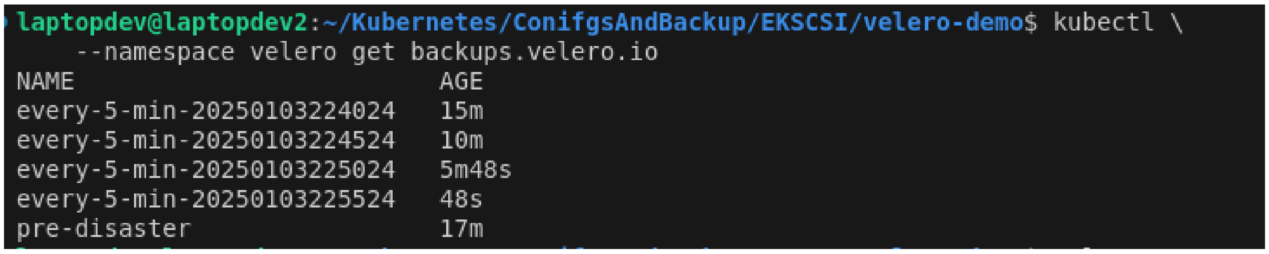
```
kubectl get namespaces
```



```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   4h20m
kube-node-lease   Active   4h20m
kube-public       Active   4h20m
kube-system       Active   4h20m
traefik           Active   4h
velero            Active   22m
```

In velero installation step, we defined s3 bucket location. We should be able to list all backups:

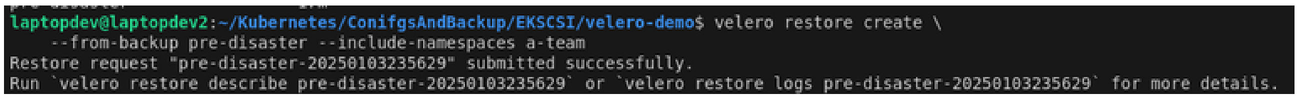
```
kubectl --namespace velero get backups.velero.io
```



```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ kubectl \
    --namespace velero get backups.velero.io
NAME                           AGE
every-5-min-20250103224024     15m
every-5-min-20250103224524     10m
every-5-min-20250103225024     5m48s
every-5-min-20250103225524     48s
pre-disaster                   17m
```

When taking backup, we will focus on just the a-team namespace by using the --include-namespaces flag.

```
velero restore create \
   --from-backup pre-disaster --include-namespaces a-team
```

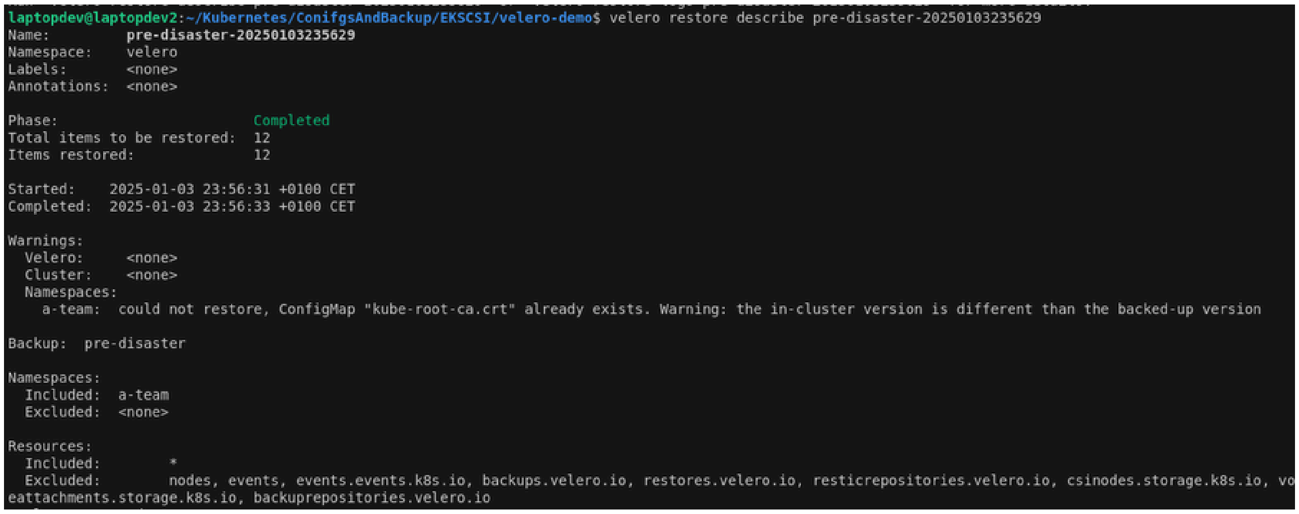


```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ velero restore create \
    --from-backup pre-disaster --include-namespaces a-team
Restore request "pre-disaster-20250103235629" submitted successfully.
Run `velero restore describe pre-disaster-20250103235629` or `velero restore logs pre-disaster-20250103235629` for more details.
```

To inspect the restore, we can run velero on the new cluster and use the describe command.

To inspect the restore, we can run velero on the new cluster and use the describe command.

```
velero restore describe <backup-name>
```

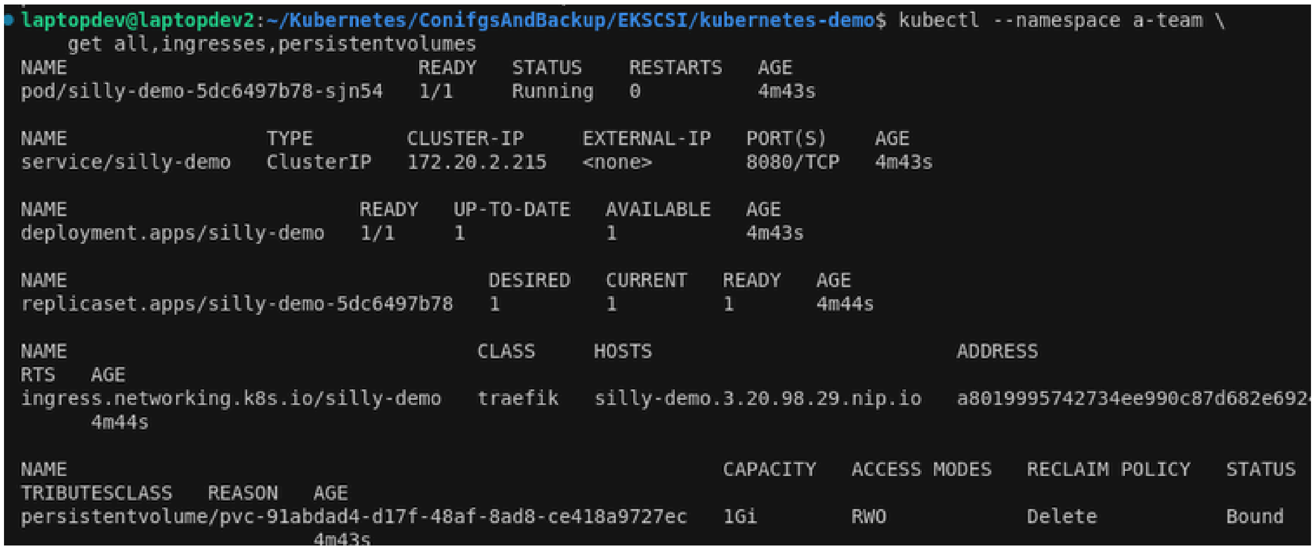


```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/velero-demo$ velero restore describe pre-disaster-20250103235629
Name:         pre-disaster-20250103235629
Namespace:    velero
Labels:       <none>
Annotations:  <none>

Phase:                       Completed
Total items to be restored:  12
Items restored:              12

Started:    2025-01-03 23:56:31 +0100 CET
Completed:  2025-01-03 23:56:33 +0100 CET

Warnings:
  Velero:     <none>
  Cluster:    <none>
  Namespaces:
    a-team:  could not restore, ConfigMap "kube-root-ca.crt" already exists. Warning: the in-cluster version is different than the backed-up version

Backup:  pre-disaster

Namespaces:
  Included:  a-team
  Excluded:  <none>

Resources:
  Included:        *
  Excluded:        nodes, events, events.events.k8s.io, backups.velero.io, restores.velero.io, resticrepositories.velero.io, csinodes.storage.k8s.io, vo
eattachments.storage.k8s.io, backuprepositories.velero.io
```

The restore completed with 12 items, including only resources from the a-team Namespace and excluding Velero-related resources.

Now check all restored resources:

```
kubectl --namespace a-team \
   get all,ingresses,persistentvolumes
```



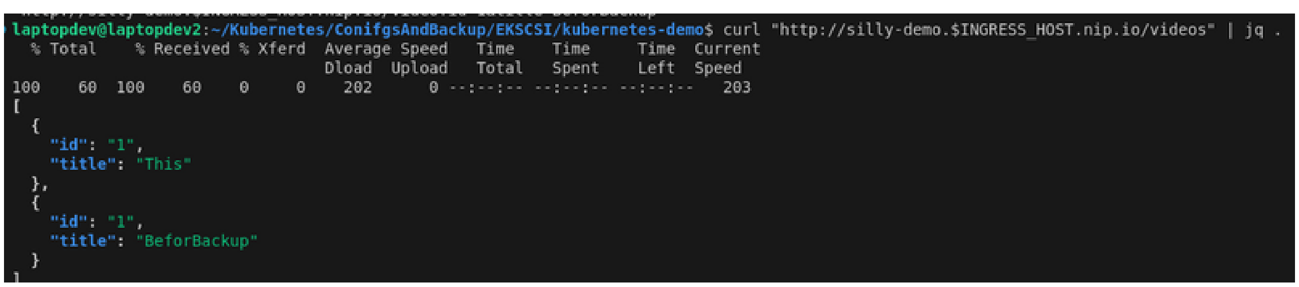
```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/kubernetes-demo$ kubectl --namespace a-team \
    get all,ingresses,persistentvolumes
 NAME                             READY   STATUS    RESTARTS   AGE
 pod/silly-demo-5dc6497b78-sjn54   1/1     Running   0          4m43s

 NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
 service/silly-demo   ClusterIP   172.20.2.215   <none>        8080/TCP   4m43s

 NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
 deployment.apps/silly-demo   1/1     1            1           4m43s

 NAME                                    DESIRED   CURRENT   READY   AGE
 replicaset.apps/silly-demo-5dc6497b78   1         1         1       4m44s

 NAME                                   CLASS     HOSTS                          ADDRESS
 RTS   AGE
 ingress.networking.k8s.io/silly-demo   traefik   silly-demo.3.20.98.29.nip.io   a8019995742734ee990c87d682e692
       4m44s

 NAME                                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
 TRIBUTESCLASS   REASON   AGE
 persistentvolume/pvc-91abdad4-d17f-48af-8ad8-ce418a9727ec   1Gi        RWO            Delete           Bound
                          4m43s
```

We confirmed Kubernetes resources were restored, but we need to check if data in external volumes was also restored. To do this, we'll send a request to the application. First, we'll update the Ingress to reflect the new load balancer's IP (or update DNS if using real domains). Follow instruction on traefik configuration from:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.
https://github.com/MichaelRobotics

now curl application and check if data in persistent volume is same as in cluster in eu-east-1:

```
curl "http://silly-demo.$INGRESS_HOST.nip.io/videos" | jq .
```



```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/kubernetes-demo$ curl "http://silly-demo.$INGRESS_HOST.nip.io/videos" | jq .
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    60  100    60    0     0    202      0 --:--:-- --:--:-- --:--:--   203
[
  {
    "id": "1",
    "title": "This"
  },
  {
    "id": "1",
    "title": "BeforBackup"
  }
]
```

Everything is working in the new cluster as if nothing happened. Both resources and external volume data were successfully restored, averting the disaster.

Kubernetes Configs: Secrets

1) Intro

Kubernetes Secrets are similar to ConfigMaps but with two key differences. First, Secrets are encrypted at rest in etcd, making them more secure for sensitive data. This encryption is typically enabled by default in managed Kubernetes, but must be manually configured in self-hosted clusters. Second, values in Secrets are base64 encoded in manifests.

example

```
git clone https://github.com/vfarcic/kubernetes-demo
cd kubernetes-demo
```

```
cat volume/config-map.yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: silly-demo
  labels:
    app.kubernetes.io/name: silly-demo
data:
  videos.yaml: |
    - id: "1"
      title: something
    - id: "2"
      title: else
    - id: "3"
      title: something new
  message: Hello, DevOps Toolkit!
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: silly-demo
  labels:
    app.kubernetes.io/name: silly-demo
[..]
          envFrom:
            - configMapRef:
                name: silly-demo
          env:
            - name: MESSAGE
              valueFrom:
                configMapKeyRef:
                  name: silly-demo
                  key: message
          volumeMounts:
            - name: cache
              mountPath: /cache
      volumes:
        - name: cache
          configMap:
            name: silly-demo
```

In this example, we define a ConfigMap called "silly-demo" with two entries: videos.yaml (YAML text) and message (plain text). Each value in a ConfigMap is always a string.

In the Deployment, we create a volume from the ConfigMap and mount it. Additionally, we can convert ConfigMap values into environment variables using envFrom. Specifically, we map the message key to an environment variable called MESSAGE.

This setup results in ConfigMap keys being stored as files, all keys converted into environment variables, and the message key as the MESSAGE environment variable

apply

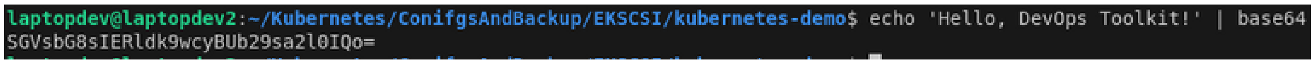
```
kubectl --namespace a-team apply \
    --filename volume/config-map.yaml
```

The Secret is similar to a ConfigMap, with the key difference that values are base64 encoded. Encoding is not encryption; it's easier to manage but not more secure.

In the Deployment, we use Secrets like ConfigMaps, specifying and mounting the silly-demo secret volume.

The value SGVsbG8sIERldk9wcyBUb29sa2l0IQo= is the base64-encoded version of "Hello, DevOps Toolkit!". You can verify this with the command echo 'Hello, DevOps Toolkit!' | base64.

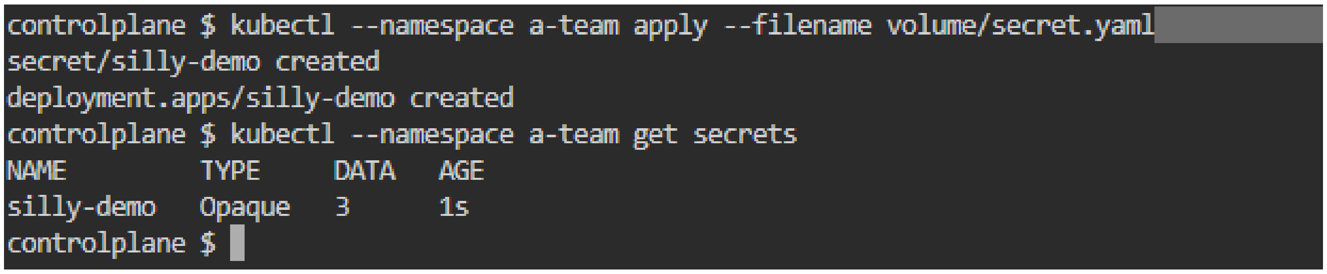
```
echo 'Hello, DevOps Toolkit!' | base64
```



```
laptopdev@laptopdev2:~/Kubernetes/ConifgsAndBackup/EKSCSI/kubernetes-demo$ echo 'Hello, DevOps Toolkit!' | base64
SGVsbG8sIERldk9wcyBUb29sa2l0IQo=
```

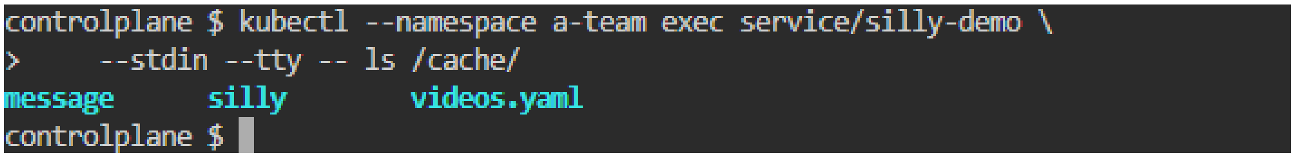
The output matches the value in the Secret manifest. Now, let's apply the new manifest to see it in action.

```
kubectl --namespace a-team apply --filename service/base.yaml
kubectl --namespace a-team apply --filename volume/secret.yaml
kubectl --namespace a-team get secrets
```



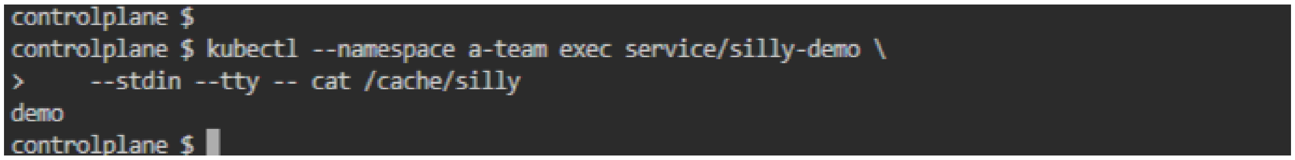
```
controlplane $ kubectl --namespace a-team apply --filename volume/secret.yaml
secret/silly-demo created
deployment.apps/silly-demo created
controlplane $ kubectl --namespace a-team get secrets
NAME         TYPE     DATA   AGE
silly-demo   Opaque   3      1s
controlplane $
```

The secret is present, and the Deployment is running. Executing into the container and listing files in the /cache directory will show the results.


```
controlplane $ kubectl --namespace a-team exec service/silly-demo \
>     --stdin --tty -- ls /cache/
message      silly        videos.yaml
controlplane $
```

Each key in the Secret has a corresponding file. We can confirm this by outputting one of the files.

```
kubectl --namespace a-team exec service/silly-demo \
   --stdin --tty -- cat /cache/silly
```


```
controlplane $
controlplane $ kubectl --namespace a-team exec service/silly-demo \
>     --stdin --tty -- cat /cache/silly
demo
controlplane $
```

Overall, Secrets are defined and used like ConfigMaps, except their values are base64 encoded and encrypted at rest.

common troubleshooting

1) ConfigMap Not Loading in Pods

Cause: Missing/misconfigured reference.
Solution: Verify ConfigMap exists (kubectl get configmap). Check Pod spec and kubectl describe pod <pod-name> for errors.

2) ETCD Backup Failing

Cause: Incorrect endpoint or permissions.
Solution: Verify ETCD endpoint and client permissions. Test manually: ETCDCTL_API=3 etcdctl snapshot save <file>.

3) Velero Backup Failing in HA EKS

Cause: IAM or S3 issues.
Solution: Check Velero IAM roles and S3 region match. Inspect logs: velero backup logs <backup-name>.

4) Secrets Not Mounting

Cause: Incorrect reference or RBAC.
Solution: Confirm Secret exists (kubectl get secret). Check Pod spec and permissions. Use kubectl describe pod <pod-name>.

5) Check my Kubernetes Troubleshooting series:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!