Backup & Restore Methods

It's important to regularly back up to ensure that your k8s cluster can be easily restored in the event of a failure or data loss. Additionally, it's important to test your backup and restore processes to ensure that they are working properly and that you can recover from any issues that may arise.

When designing a backup strategy for a Kubernetes cluster, it's crucial to back up both the application data and the cluster configuration.

Cluster configuration

Cluster configuration includes all the Kubernetes objects and resources that configure your cluster and applications

etcd data: The cluster state and metadata in Kubernetes are stored in etcd. To ensure cluster recovery, it's crucial to back up the etcd data. This can be achieved either by taking periodic snapshots of the etcd database or by implementing a backup solution specifically designed for etcd, such as etcdctl or Velero.

Kubernetes manifests: includes all the Kubernetes objects and resources that configure your cluster and applications. This includes things like deployments, services, configmaps, and etc.These resources are usually defined as code, for example in YAML or JSON files. Because they are code, a good practice is to store them in a version control system like Git. This gives you a history of changes and allows you to revert to a previous state if something goes wrong.

You can backup Kubernetes resources using etcdctl command-line

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: etcd-backup
spec:
  schedule: "0 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: etcd-backup
            image: quay.io/coreos/etcd:v3.5.0
            command:
            - /bin/sh
            - -c
            - |
              ETCDCTL_API=3 etcdctl snapshot save /backup/k8s/etcd-snapshot.db \
              --endpoints=<ETCD-endpoints> \
              --cacert=/etc/kubernetes/pki/etcd/ca.crt \
              --cert=/etc/kubernetes/pki/etcd/server.crt \
              --key=/etc/kubernetes/pki/etcd/server.key
            volumeMounts:
            - name: etcd-certs
              mountPath: /etc/kubernetes/pki/etcd
            - name: backup
              mountPath: /backup
          volumes:
          - name: etcd-certs
            secret:
              secretName: etcd-certs
          - name: backup
            persistentVolumeClaim:
              claimName: backup-pvc
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: backup-pvc
spec:
  storageClassName: <storage-class>
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

This manifest sets up a CronJob that runs an etcd backup job every hour, using the etcdctl command-line tool inside a container to create a snapshot of the etcd database and save it to the specified path. It mounts the etcd certificates and a PersistentVolumeClaim for storing the backup.

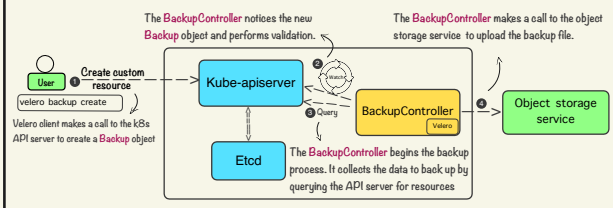
You can backup Kubernetes resources using Velero

Once Velero is installed, you can create a backup by running the following command:

```
velero backup create <backup-name>
```

By default, Velero will back up all resources in all namespaces. If you want to back up only certain namespaces or resources, you can specify them with the --include-namespaces and --include-resources flags, respectively

```
velero backup create <backup-name> --include-namespaces my-namespace \
--include-resources deployments,pods
```



To restore a backup, run the following command:

```
velero restore create --from-backup <backup-name>
```

By default, Velero will restore all resources in the backup to their original namespaces. If you want to restore only certain namespaces or resources, you can specify them with the --include-namespaces and --include-resources flags, respectively

```
velero restore create --from-backup <backup-name> --include-namespaces my-namespace \
--include-resources deployments,pods
```

you can also automate the backup process with Velero

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: velero-backup
spec:
  schedule: "0 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: velero
            image: velero/velero:v1.7.0
            command:
            - /velero
            args:
            - backup
            - create
            - my-backup
            volumeMounts:
            - name: cloud-credentials
              mountPath: /credentials
            env:
            - name: AWS_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: cloud-credentials
                  key: aws_access_key_id
            - name: AWS_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: cloud-credentials
                  key: aws_secret_access_key
          volumes:
          - name: cloud-credentials
            secret:
              secretName: cloud-credentials
```

To restore an etcd backup using etcdctl

1. Ensure that the Kubernetes API server is not active or stopped

```
sudo mv /etc/kubernetes/manifests/kube-api.yaml another-path
```

Or

```
sudo systemctl stop kube-apiserver
```

2. change the path of the ETCD data directory to var /var/lib/etcd-from-backup/, you need to edit the manifest file and update the relevant volume and hostPath specifications

```
/etc/kubernetes/manifests/etcd.yaml
volumes:
- name: etcd-data
  hostPath:
    path: /var/lib/etcd-from-backup/
```

3. Use the etcdctl to restore the backup

```
etcdctl snapshot restore /backup/k8s/etcd-snapshot.db \
--data-dir=/var/lib/etcd-from-backup \
--initial-cluster= etcd01= kubemaster-1=https://192.168.100.11:2380 \
--initial-advertise-peer-urls https://192.168.100.11:2380 \
--initial-cluster-token=<ETCD-initial-cluster-token> \
--name=kubemaster-1
```

Application data

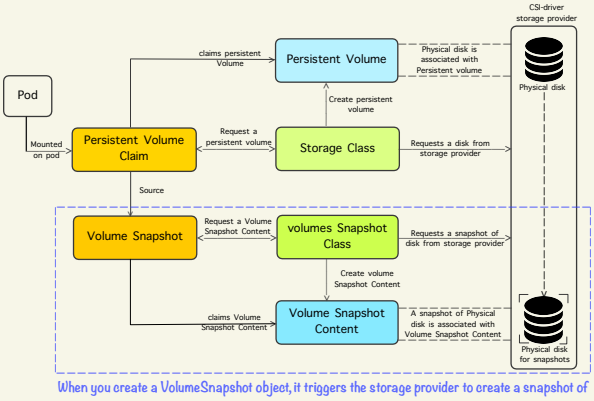
Refers to the actual data produced and managed by the applications running on your k8s cluster. This could include databases, user-generated content, logs, and anything else that your applications are producing or manipulating

There are several strategies you can follow to backup this data:

Database Backups: If you're using a database in your application, it's likely that the database itself has backup functionality. For example, you can create a dump of a MySQL database or a snapshot of a MongoDB database.

Backup Sidecars: Another approach is to use a sidecar container in your pods specifically for managing backups. This container would be responsible for regularly creating backups and sending them to a remote location

Volume Snapshots: Kubernetes volume snapshots provide a standardized way to create copies of the content of persistent volumes at a point in time, without creating new volumes.



When you create a VolumeSnapshot object, it triggers the storage provider to create a snapshot of the underlying storage volume. The snapshot is represented by the VolumeSnapshotContent object

To create a VolumeSnapshot in Kubernetes, follow the steps below

Ensure that you have the necessary prerequisites in place

Cluster must have volume snapshot CRDs, and snapshot controller deployed on it for this to work

```
kubectl get crds | grep snapshot.storage.k8s.io
```

CSI driver and storage class that support volume snapshots

Create a VolumeSnapshot of the desired PVC

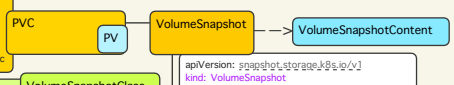
Define a VolumeSnapshot object that references the PVC you want to snapshot

Verify VolumeSnapshotContent was created

Check the status of the VolumeSnapshot to ensure it is created successfully

```
kubectl describe volumesnapshot <snapshot-name>
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-hostpath-sc
```



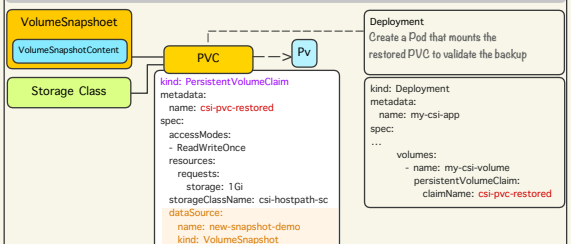
When a VolumeSnapshot object is created, the VolumeSnapshotClass provisions a VolumeSnapshotContent to hold the actual snapshot data.. Deleting the VolumeSnapshot object does not delete the VolumeSnapshotContent object. If you want to delete the snapshot data, you need to delete the corresponding VolumeSnapshotContent object

```yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-sc
driver: hostpath.csi.k8s.io
deletionPolicy: Delete
```

```yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: new-snapshot-demo
spec:
  volumeSnapshotClassName: csi-hostpath-sc
  source:
    persistentVolumeClaimName: csi-pvc
```

The VolumeSnapshotClass defines the snapshotter/provisioner that will be used to take snapshots and parameters like retention policy, etc. it enables dynamic provisioning of snapshots, just like a StorageClass allows dynamic provisioning of volumes

To restore a snapshot, create a new PVC based on a VolumeSnapshotContent. This results in a new PV with data populated from the snapshot




```yaml
kind: PersistentVolumeClaim
metadata:
  name: csi-pvc-restored
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-hostpath-sc
  dataSource:
    name: new-snapshot-demo
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

New PV provisioned from the snapshot data

```yaml
kind: Deployment
metadata:
  name: my-csi-app
spec:
  ...
    volumes:
      - name: my-csi-volume
        persistentVolumeClaim:
          claimName: csi-pvc-restored
```

The dataSource field indicates that this PVC is a clone of the specified snapshot

Security

Kubernetes uses a combination of secure network channels, authentication and authorization mechanisms, network policies, and container security features to ensure that all communication within the cluster is authenticated, encrypted, and secure. These mechanisms help to protect the cluster against unauthorized access, data breaches, and other security threats, and provide a reliable and secure platform for deploying and managing containerized applications.

Secure network channels: Kubernetes uses secure network channels to ensure that all communication within the cluster is encrypted and secure. These channels are established using Transport Layer Security (TLS) certificates, which provide a secure way to authenticate the identity of different components and encrypt all data that is transmitted between them.

In Kubernetes, many of the components use mutual TLS (Transport Layer Security) authentication for secure communication between each other. This method involves each component having its own certificate (crt) and private key (key) that are used to authenticate and encrypt communication when communicating with other components.

The cluster's certificate authority (CA) is responsible for issuing and managing certificates used for authentication and encryption within the cluster. the CA is typically implemented as a component within the Kubernetes control plane, and is responsible for generating and managing the cluster's root certificate and private key. These are used to sign and issue certificates for different components within the cluster, such as nodes, API servers, and users.

/etc/kubernetes/ is a directory that contains Kubernetes configuration files. usually used for defining settings related to the k8s components

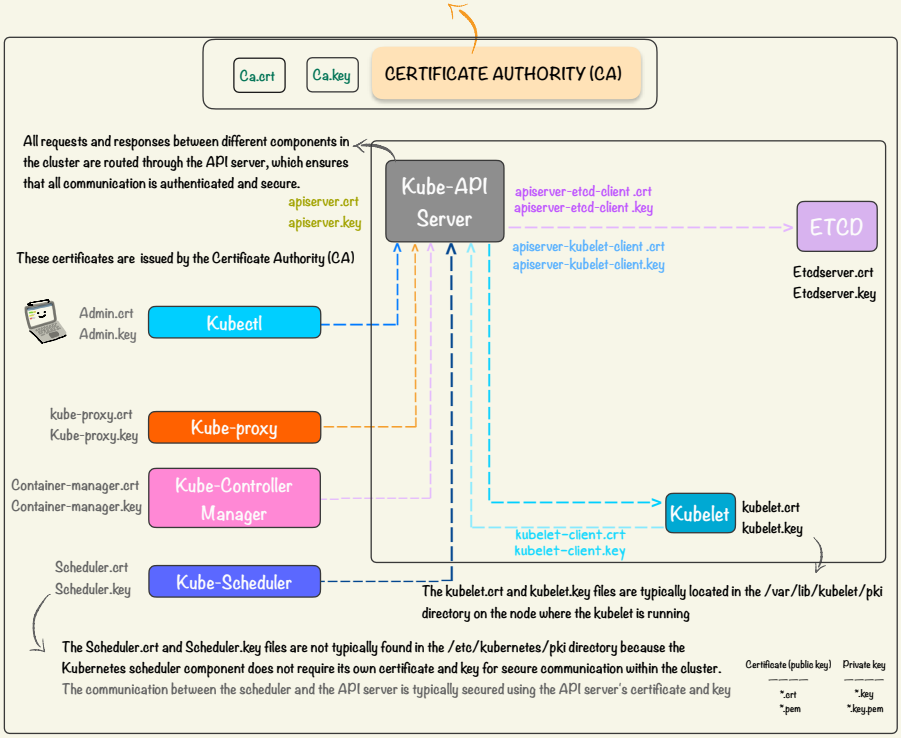
```
arye@kubemaster-1:/etc/kubernetes$ ll
-rw-------  1 root root 5450 May 18 09:34 admin.conf
-rw-------  1 root root 5486 May 18 09:34 controller-manager.conf
-rw-------  1 root root 1886 May 18 09:35 kubelet.conf
drwxr-xr-x  2 root root 4096 May 18 10:30 manifests/
drwxr-xr-x  3 root root 4096 May 18 09:34 pki/
-rw-------  1 root root 5438 May 18 09:34 scheduler.conf
```

These configuration files are essential for the proper functioning of the various k8s components. They contain settings such as the API server address, authentication and authorization information, and other component-specific configurations

admin.conf: This file contains the configuration for the Kubernetes cluster administrator, holding the necessary credentials and cluster information to interact with the cluster using the kubectl command-line tool

The /etc/kubernetes/pki directory is a directory used by Kubernetes to store the public key infrastructure (PKI) materials, such as certificates and keys, that are used to secure communication between the different components of the Kubernetes cluster.

```
arye@kubemaster-1:/etc/kubernetes/pki$ ll
-rw-r--r-- 1 root root 1090 May 18 09:34 apiserver-etcd-client.crt
-rw------- 1 root root 1679 May 18 09:34 apiserver-etcd-client.key
-rw-r--r-- 1 root root 1099 May 18 09:34 apiserver-kubelet-client.crt
-rw------- 1 root root 1675 May 18 09:34 apiserver-kubelet-client.key
-rw-r--r-- 1 root root 1229 May 18 09:34 apiserver.crt
-rw------- 1 root root 1679 May 18 09:34 apiserver.key
-rw-r--r-- 1 root root 1025 May 18 09:34 ca.crt
-rw------- 1 root root 1679 May 18 09:34 ca.key
drwxr-xr-x 2 root root 4096 May 18 09:34 etcd/
-rw-r--r-- 1 root root 1038 May 18 09:34 front-proxy-ca.crt
-rw------- 1 root root 1679 May 18 09:34 front-proxy-ca.key
-rw-r--r-- 1 root root 1058 May 18 09:34 front-proxy-client.crt
-rw------- 1 root root 1675 May 18 09:34 front-proxy-client.key
-rw------- 1 root root 1675 May 18 09:34 sa.key
-rw------- 1 root root  451 May 18 09:34 sa.pub
```



Certificates generated by kubeadm expire after 1 year and will need to be renewed. kubeadm provides a simple command to renew all certificates

To renew all the certificates in a k8s cluster with kubeadm, you can use the kubeadm certs renew command with the all option

```
root@kubemaster-1 ( /etc/kubernetes):
 kubeadm certs renew all
```

This will renew the following certificates:
- etcd server and peer certificates
- API server certificate
- Front proxy client certificate
- Controller manager client certificate
- Scheduler client certificate

Note: kubelet.conf is not included in the list above

you can check the expiration dates of the certificates

```
arye@kubemaster-1:~$ sudo kubeadm certs check-expiration
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE AUTHORITY   EXTERNALLY MANAGED
admin.conf                 May 18, 2023 09:34 UTC   341d                                    no
apiserver                  May 18, 2023 09:34 UTC   341d            ca                      no
apiserver-etcd-client      May 18, 2023 09:34 UTC   341d            etcd-ca                 no
apiserver-kubelet-client   May 18, 2023 09:34 UTC   341d            ca                      no
controller-manager.conf    May 18, 2023 09:34 UTC   341d                                    no
etcd-healthcheck-client    May 18, 2023 09:34 UTC   341d            etcd-ca                 no
etcd-peer                  May 18, 2023 09:34 UTC   341d            etcd-ca                 no
etcd-server                May 18, 2023 09:34 UTC   341d            etcd-ca                 no
front-proxy-client         May 18, 2023 09:34 UTC   341d            front-proxy-ca          no
scheduler.conf             May 18, 2023 09:34 UTC   341d                                    no

CERTIFICATE AUTHORITY      EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                         May 15, 2032 09:34 UTC   9y              no
etcd-ca                    May 15, 2032 09:34 UTC   9y              no
front-proxy-ca             May 15, 2032 09:34 UTC   9y              no
```

you can use the following command to display the details of a certificate file in a human-readable format:

```
openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text -noout
```

It is advisable to backup your certificates and configuration files before executing the command

/etc/kubernetes/pki/*.* | /etc/kubernetes/*.conf | ~/.kube/config

After running the command you should restart the control plane Pods

Static Pods are managed by the local kubelet and not by the API Server, thus kubectl cannot be used to delete and restart them. To restart a static Pod you can temporarily remove its manifest file from /etc/kubernetes/manifests/

kubeadm can renews all the certificates during control plane upgrade.

This feature is intended to address straightforward scenarios. If you don't have specific requirements regarding certificate renewal and regularly perform Kubernetes version upgrades (with less than a year between each upgrade), kubeadm will handle the process to ensure your cluster remains up to date and reasonably secure.