VERITAS

Veritas InfoScale™Disaster Recovery Configuration with Encrypted Data in Kubernetes Environment

Contents

Executive Summary

Infoscale supports containerization using the platforms like Kubernetes and OpenShift. The use of Infoscale in Kubernetes helps customers to easily manage their containers, automate deployment processes, and scale their applications as needed, thus providing the reliability and scalability that businesses need to succeed in today's fast-paced digital landscape. Apart from Disaster Recovery, Infoscale also provide functionalities like Fencing, Licensing, Snapshots, Clones, VVR, CSI, DR for Cloud, etc. Some of the disaster recovery features in Infoscale for Kubernetes include:

- Data Backup and Recovery: Infoscale provides automated backup and recovery options to help customers protect their data and applications.
- High Availability: Infoscale provides high availability features such as automatic failover and self-healing capabilities, to help ensure that applications continue to run even in the event of hardware failures or other disruptions.

In Infoscale DR, data can be securely sent in encrypted format by using keys acquired from KMS server. This paper is focussed on KMS server-side configuration and Infoscale DR client-side configuration.

Infoscale for Kubernetes

Infoscale provides high-performance shared storage for the OpenShift or Kubernetes clusters by using the fast storage, which is directly attached to the cluster nodes. Infoscale Storage provides highly available persistent storage that conforms to CSI specifications for enterprise applications by using high-performance parallel storage access on shared storage (SAN)or in Flexible Storage Sharing (FSS)environments.

DR in Infoscale for Kubernetes

Disaster recovery refers to the process of restoring an organization’s systems and data in the event of a natural or man-made disaster. This includes restoring hardware, software, and data from backups, as well as implementing measures to prevent data loss and minimize downtime. One key component of disaster recovery is data backup and recovery. Another important aspect of disaster recovery is ensuring that systems can be quickly and easily restored in the event of a disaster. Infoscale in Kubernetes environment achieves this with the help of a DR Manager, which orchestrates migration and takeover flow and provideuseful messages/errors related to the communication between the two sites. The DR in Infoscale works as below:

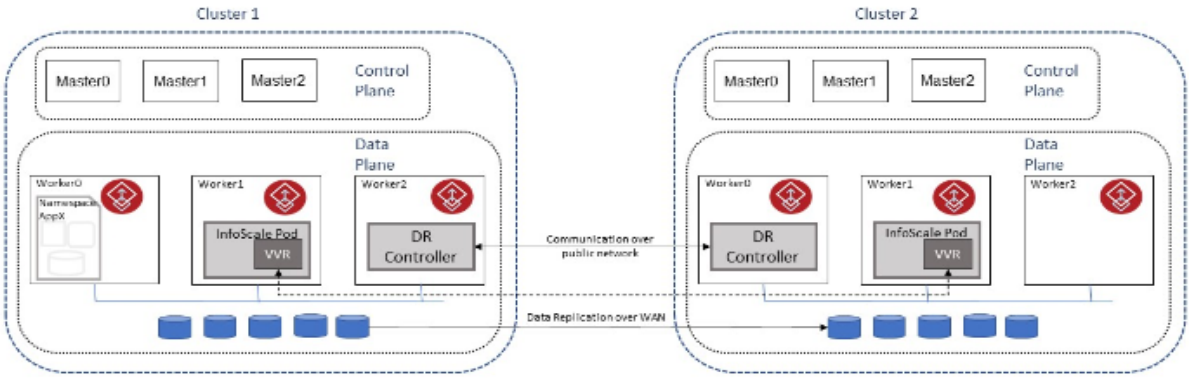


Fig 1: DR Configuration Between two clusters.

Data Encryption in Infoscale DR for Kubernetes

In infoscale DR, encryption is not enabled by default. But when encryption is enabled, user data is encrypted by using symmetric key encryption and sent in an encrypted format. These keys are acquired from the KMS server.

About the Custom CA Configuration on Kubernetes Cluster

Certificate Authority (CA) is a trusted entity that issues the SSL certificates. Infoscale cluster can be configured by using a self signed issuer or custom CA. When data encryption is enabled, Infoscale cluster need to be configured with custom CA.

In this document Kubernetes certificate and key is used as root authority for custom CA configuration. Any other tool can be used to configure the custom CA.

Pre-requisites:

1. Download tools cfssl and cfssljson. Run the following commands.
   go get / install github.com/cloudflare/cfssl/cmd/cfssl
   go get / install github.com/cloudflare/cfssl/cmd/cfssljson
2. Use certificate and key from following path on Kubernetes cluster.
   /etc/kubernetes/pki/ca.crt
   /etc/kubernetes/pki/ca.key
3. Create vxconfig.json file with the following contents.

```
{
  "signing": {
        "default": {
          "expiry": "43800h"
        },
        "profiles": {
          "cluster": {
                "expiry": "8760h",
                "usages": [
                  "signing",
                  "key encipherment",
                  "cert sign",
                  "server auth",
                  "client auth"
                ],
                "ca_constraint": {
                  "is_ca": true
                }
          }
        }
  }
}
```

4. Create csr_config.json file with the following contents

```
{
  "CN": "infoscale-ca",
  "key": {
        "algo": "rsa",
        "size": 2048
  },
  "hosts": [
        "kubernetes"
  ],
  "names": [
        {
          "O": "system:nodes",
          "OU": "vx"
        }
  ]
}
```

Step 1: Generate Infoscale-ca CSR and Key as under

```
[root@r7515-046-vm4 sec]#
[root@r7515-046-vm4 sec]# cfssl genkey csr_config.json | cfssljson -bare infoscale-ca
2023/02/26 13:32:04 [INFO] generate received request
2023/02/26 13:32:04 [INFO] received CSR
2023/02/26 13:32:04 [INFO] generating key: rsa-2048
2023/02/26 13:32:04 [INFO] encoded CSR
[root@r7515-046-vm4 sec]# ls
csr_config.json  infoscale-ca.csr  infoscale-ca-key.pem  vxconfig.json
[root@r7515-046-vm4 sec]#
```

Step 2: Generate Infoscale-ca Certificate as under

```
[root@r7515-046-vm4 sec]#
[root@r7515-046-vm4 sec]# cfssl sign -ca /etc/kubernetes/pki/ca.crt -ca-key /etc/kubernetes/pki/ca.key -hostname kubernetes
 -config ./vxconfig.json -profile cluster ./infoscale-ca.csr  | cfssljson -bare infoscale-ca
2023/02/26 13:42:58 [INFO] signed certificate with serial number 305514960318951805046437060252269398342655257281
[root@r7515-046-vm4 sec]# ls
csr_config.json  infoscale-ca.csr  infoscale-ca-key.pem  infoscale-ca.pem  vxconfig.json
[root@r7515-046-vm4 sec]#
```

Step 3: Create Infoscale-ca secret file as under

```
[root@r7515-046-vm4 sec]# cat <<EOF > custom-ca.yaml
> ---
> apiVersion: v1
> kind: Namespace
> metadata:
>   labels:
>     control-plane: infoscale-sds-operator
>   name: infoscale-vtas
> ---
> apiVersion: v1
> kind: Secret
> metadata:
>   name: infoscale-ca
>   namespace: infoscale-vtas
> type: kubernetes.io/tls
> data:
>   ca.crt: $(kubectl get cm kube-root-ca.crt -o jsonpath="{.data['ca\.crt']}" | base64 -w0)
>   tls.crt: $(base64 ./infoscale-ca.pem | tr -d '\n')
>   tls.key: $(base64 ./infoscale-ca-key.pem | tr -d '\n')
> EOF
[root@r7515-046-vm4 sec]# ls
csr_config.json  custom-ca.yaml  infoscale-ca.csr  infoscale-ca-key.pem  infoscale-ca.pem  vxconfig.json
```

Step 4: Apply custom-ca.yaml created in above step on all member clusters which are a part of DR and verify Infoscale-ca secret is created

```
[root@r7515-046-vm4 sec]# kubectl -n infoscale-vtas get secret
NAME                                TYPE                DATA   AGE
infoscale-ca                        kubernetes.io/tls   3      6d17h
```

Infoscale Configuration on Kubernetes

Configure Infoscale cluster on all Kubernetes clusters which are a part of DR.

Infoscale cluster can be configured by applying yamls like sro, sr, lico, license_cr, iso, cr in the same sequence. For more details, see:
https://sort.veritas.com/documents/doc_details/IEK/8.0.200/General/Documentation/

```
[root@r7515-046-vm4 ~]# kubectl -n infoscale-vtas get pods
NAME                                              READY   STATUS    RESTARTS       AGE
infoscale-csi-controller-32414-0                  5/5     Running   50 (42h ago)   5d9h
infoscale-csi-node-32414-5rpf5                    2/2     Running   22 (42h ago)   6d23h
infoscale-csi-node-32414-bckkc                    2/2     Running   22 (42h ago)   6d23h
infoscale-csi-node-32414-qhj9q                    2/2     Running   22 (42h ago)   6d23h
infoscale-dr-manager-57d9f9588-hx5j5              1/1     Running   0              42h
infoscale-fencing-controller-32414-5d8455587-clgwr   1/1     Running   0              42h
infoscale-fencing-enabler-32414-4f7q4             1/1     Running   21 (42h ago)   6d23h
infoscale-fencing-enabler-32414-ktvhn             1/1     Running   21 (42h ago)   6d23h
infoscale-fencing-enabler-32414-sh6zg             1/1     Running   22 (42h ago)   6d23h
infoscale-licensing-operator-d4f6f5bb9-kd6qb      1/1     Running   11 (42h ago)   6d23h
infoscale-sds-32414-5cad07c8bf71158c-97gzx        1/1     Running   11 (42h ago)   6d23h
infoscale-sds-32414-5cad07c8bf71158c-jp6ng        1/1     Running   11 (42h ago)   6d23h
infoscale-sds-32414-5cad07c8bf71158c-pcjbc        1/1     Running   11 (42h ago)   6d23h
infoscale-sds-operator-54496584db-h475l           1/1     Running   0              42h
```

About the KMIP Secret Configuration

KMIP Secret contains the KMS server and port information. Using this secret the cluster can be connected with the KMS server.

Pre-requisites:

1. KMS server must be pre-configured and KMS server hostname or IP address and port details must be ready
2. The KMS server hostname must be Base64 encoded.
3. the KMS server port must be Base64 encoded.
4. infoscale-kmip-encrypt.yaml must be created by using encoded hostname and port.

```
[root@r7515-046-vm4 sec]#
[root@r7515-046-vm4 sec]# cat infoscale-kmip-encrypt.yaml
apiVersion: v1

data:

  host: MTAuMjEwLjE4Mi4xOTgK

  port: NTY5Ngo=

kind: Secret

metadata:

  name: infoscale-kmip-encrypt

  namespace: infoscale-vtas

type: Opaque
```

Step 1: Apply infoscale-kmip-encrypt.yaml created in above step on all member clusters which are a part of DR and verify infoscale-kmip-encrypt is created.

```
[root@r7515-046-vm4 sec]#
[root@r7515-046-vm4 sec]# kubectl -n infoscale-vtas get secret | grep kmip
infoscale-kmip-encrypt             Opaque              2      6d17h
[root@r7515-046-vm4 sec]#
```

About the KMS Server Certificate Configuration

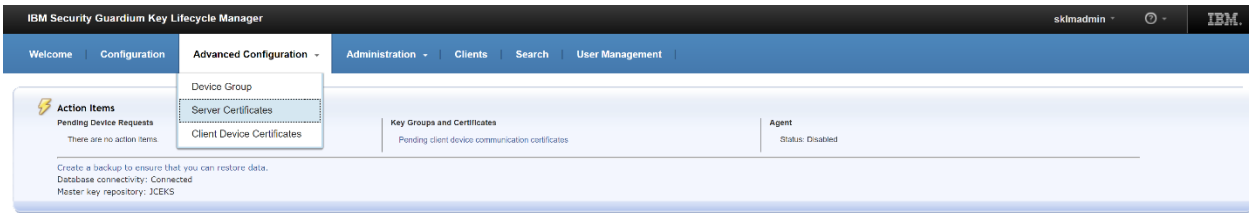
Create server certificate on KMS server. Client clusters can connect to the server by using this server certificate.

Pre-requisites:

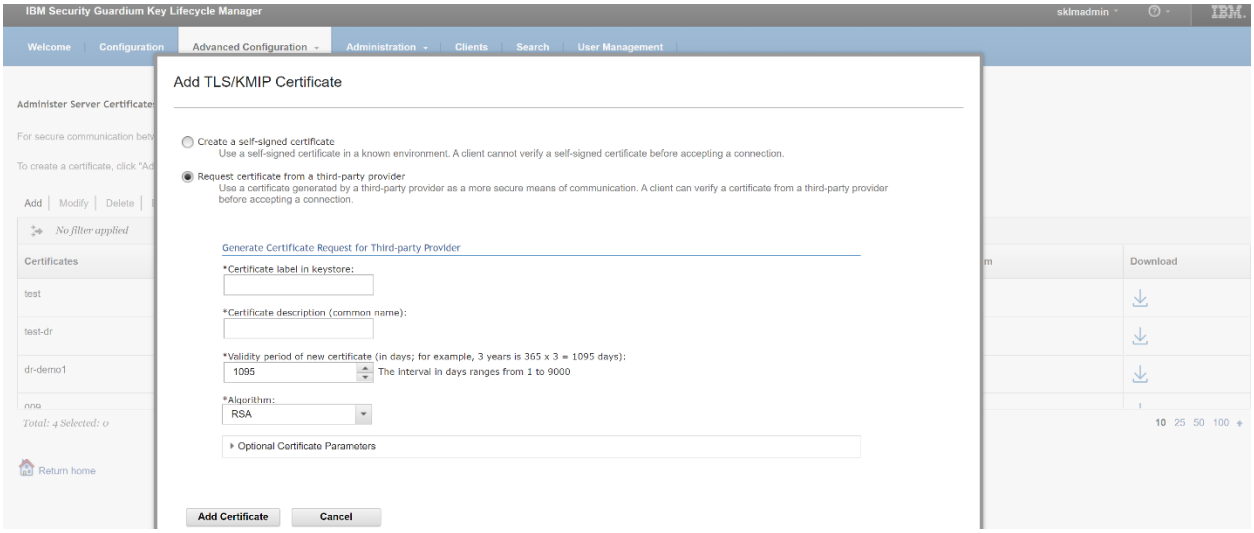
1. KMS web server must be accessible with the given credentials and port details.

Step 1: Login to IBM security key lifecycle manager (https://<kms-server>:<port>/ibm/SKLM/login.jsp)

Step 2: Go to Advance Configuration -> Server Certificate



Step 3: Go to Add -> Request Certificate from a third party



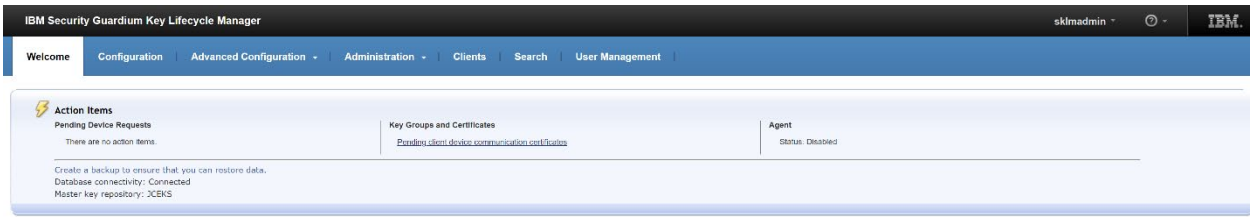
Certificate $certificate-label.csr is generated at /opt/IBM/WebSphere/AppServer/products/sklm/ on the KMS server

Step 4: Copy $certificate-label.csr from the KMS server to the local kubernetes cluster.

Step 5: Sign the CSR on the kubernetes cluster to generate the server certificate.

```
[root@r7515-046-vm4 sec]#
[root@r7515-046-vm4 sec]# openssl x509 -req -in 220817103912-test-dr.csr -CA infoscale-ca.pem
    -CAkey infoscale-ca-key.pem -CAcreateserial -out server-cert.pem -days 1024 -sha256
Signature ok
subject=CN = test-dr
Getting CA Private Key
[root@r7515-046-vm4 sec]#
```

Step 6: Web login KMS Server -> Welcome -> Third party certificates pending import.



Step 7: Click on Server Certificate and import server-cert.pem

Step 8: Go to Advance Configuration -> Server Certificate -> Click on server-cert -> Modify -> Enable check box for the current certificate.

About the Client Certificates Configuration

Client certificate configuration includes get client certificate from the infoscale cluster, register certificate on KMS server and then create a client group. With this configuration, multiple clusters can use the same certificates while configuring encrypted volumes.

Pre-requisites:

1. Infoscale cluster must be configured to generate the client certificate.
2. KMS web server must be accessible with the given credentials and port details

Step 1: Get client certificate from all member clusters which are a part of DR

Client certificate from client-one

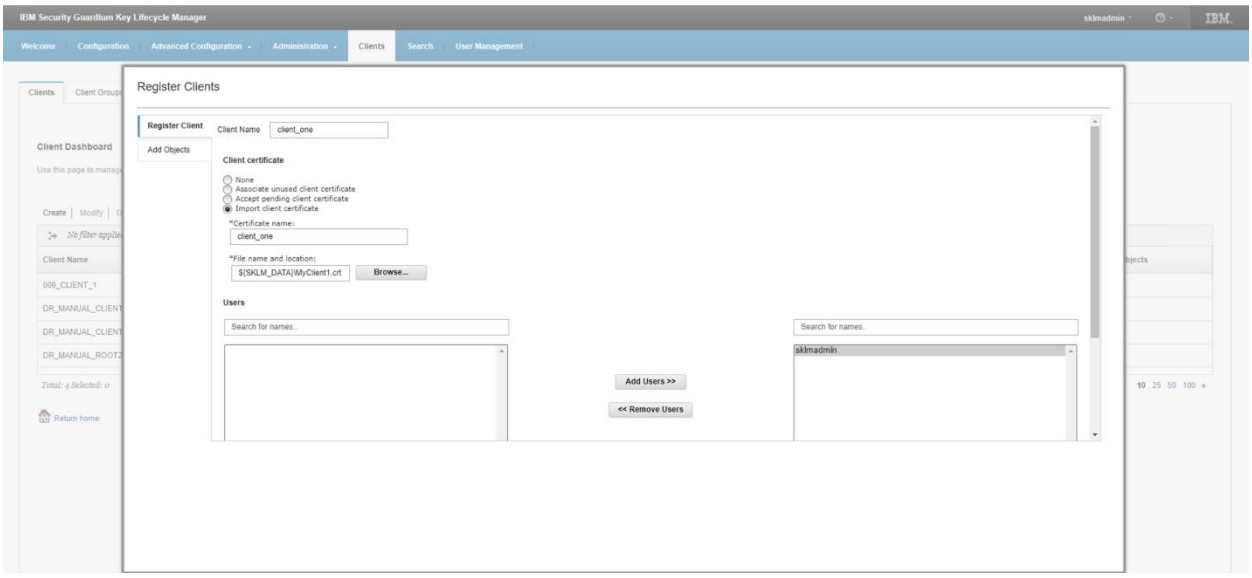
```
[root@r7515-046-vm4 ~]#
[root@r7515-046-vm4 ~]# kubectl -n infoscale-vtas get secret infoscale-sds-rest-tls-cert-32414
         -o jsonpath="{.data['tls\.crt']}" | base64 --decode > kmip-client-one-cert.crt
[root@r7515-046-vm4 ~]# ls kmip-client-one-cert.crt
kmip-client-one-cert.crt
[root@r7515-046-vm4 ~]#
```

Client certificate from client-two

```
[root@r7515-046-vm9 sec]#
[root@r7515-046-vm9 sec]# kubectl -n infoscale-vtas get secret infoscale-sds-rest-tls-cert-32415
           -o jsonpath="{.data['tls\.crt']}" | base64 --decode > kmip-client-twi-cert.crt
[root@r7515-046-vm9 sec]# ls kmip-client-twi-cert.crt
kmip-client-twi-cert.crt
[root@r7515-046-vm9 sec]#
```

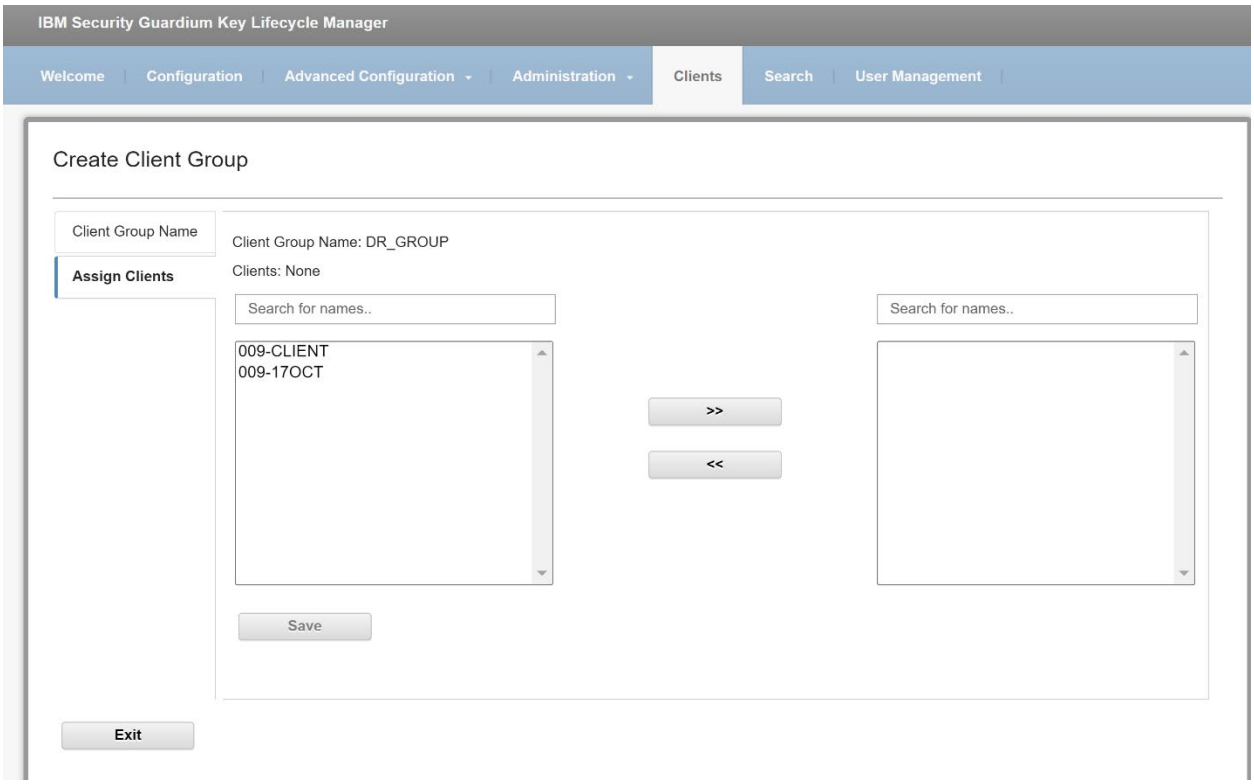
Step 2: Register clients on KMS server using client certificates

Login to KMS server web console -> Clients -> Create -> Add client certificate details -> Register Client



Step 3: Create Client group on KMS server for all registered clients

Login to KMS server web console -> Clients->Client Groups->Add->Client Group Name (Specify client group name)->Assign Clients (Specify clients which you want to be a part of group)



Step 4: Must be able to create PVC on all client clusters by using same certificate from KMS server with encrypted storage class

Perform following test to validate if DR can be configured for encrypted PVC.

- Apply encrypted storage class on all the clusters

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-infoscale-sc-encrypted
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: org.veritas.infoscale
reclaimPolicy: Delete
allowVolumeExpansion: true
parameters:
  fstype: vxfs
  layout: "mirror"
  faultTolerance:: "2"
  mediaType: "hdd"
  encryption: "true"
```

- Apply PVC with encrypted storage class on all the clusters

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mysql-pvc-app1-enc
  namespace: mysql
  labels:
    env: app1
spec:
  storageClassName: csi-infoscale-sc-encrypted
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

- Must be able to configure PVC on all client clusters by using certificate from the same KMS server.

```
[root@r7515-046-vm4 sec]#
[root@r7515-046-vm4 sec]# kubectl -n mysql get pvc
NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS               AGE
mysql-pvc-app1-enc   Bound    pvc-c5932cb7-2277-49b4-95c4-31329bd98c07   1Gi        RWX            csi-infoscale-sc-encrypted   5m56s
[root@r7515-046-vm4 sec]#
```

```
[root@r7515-046-vm9 sec]#
[root@r7515-046-vm9 sec]# kubectl -n mysql get pvc
NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS               AGE
mysql-pvc-app1-enc   Bound    pvc-c5932cb7-2277-49b4-95c4-31329bd98c07   1Gi        RWX            csi-infoscale-sc-encrypted   3m22s
[root@r7515-046-vm9 sec]#
```

Configure Application on Encrypted Storage Class

Configure PostresSQL application on one of the Kubernetes clusters by using storage from encrypted storage class. The cluster site where application is configured act as a primary site whereas other clusters where application is not running act as secondary sites.

Step 1: Apply encrypted storage class on all clusters which are a part of DR.

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-infoscale-sc-encrypted
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: org.veritas.infoscale
reclaimPolicy: Delete
allowVolumeExpansion: true
parameters:
  fstype: vxfs
  layout: "mirror"
  faultTolerance:: "2"
  mediaType: "hdd"
  encryption: "true"
```

Step 2: Apply PVC with encrypted storage class on the primary site.

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: postgres-pvc-app1-enc
  namespace: postgressql
  labels:
    env: app1
spec:
  storageClassName: csi-infoscale-sc-encrypted
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

```
[root@r7515-046-vm4 sec]# kubectl -n postgressql get pvc
NAME                 STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS                AGE
postgres-pvc-app1-enc   Bound    pvc-903ed67e-9ba3-4b9d-a790-217bb1a4a154   1Gi        RWX            csi-infoscale-sc-encrypted   50s
[root@r7515-046-vm4 sec]#
```

Step 3: Apply PostgresSQL application on the primary site.

```
[root@r7515-046-vm4 sec]#
[root@r7515-046-vm4 sec]# kubectl -n postgressql get pods
NAME                                READY   STATUS    RESTARTS   AGE
postgressql-app1-f654bbd77-g4qn7    1/1     Running   0          5s
[root@r7515-046-vm4 sec]#
```

Configure Application DR

DR can be configured for application by applying following custom resource

- Configure GCM: Cluster membership configured between clusters in DR
- Configure Datareplication: VVR replication configured between clusters in DR
- Configure Disasterrecoveryplans: DR operations can trigger using drplan

For more details, see:
https://sort.veritas.com/documents/doc_details/IEK/8.0.200/General/Documentation/

Conclusion

With encryption we can ensure data security at rest and during transmission and disaster recovery provide high availability for application in the event of cluster failure. This document discusses how DR can achieved for application configured on persistent storage from encrypted storage class and using IBM KMS certificates for encryption.

References

1. Veritas Infoscale for Kubernetes Guide for version 8.0.200:
   https://sort.veritas.com/documents/doc_details/IEK/8.0.200/General/Documentation/

2. Kubernetes documentation
   https://kubernetes.io/docs/