

Maintaining your KumoScale Deployment

This section presents information on various activities you may find helpful when developing and supporting an implementation using KumoScale software.

Topics on this page

- [Failure Recovery](#)
- [Planned Maintenance](#)
- [Using PEM Files](#)
- [Replacing the Provisioner key](#)
- [Modifying secrets](#)
- [Replacing a License](#)
- [Initiator NQN Identifier](#)
- [Field Types](#)
- [Rebooting a storage node](#)
- [Restore factory settings \(Appliance only\)](#)
- [Generating Log Files](#)
- [Listing Tasks](#)
- [Locating an SSD](#)
- [Multipathing Support](#)

Failure Recovery

Replicas may go down temporarily because of a network failure or a maintenance operation over the rack. In this case, the orchestrator will migrate the container to a node on another rack containing one of the other replicas of the volume and reconnect to it.

The KumoScale CSI driver monitors the mdraid[1] (MD) of each volume every 30 seconds and repairs inconsistencies and incomplete states. In case of a failure, KumoScale initiates a self-healing process that queries the KumoScale Provisioner Service and performs the following tasks:

- If an application initiator was connected to a target, it reconnects it.
- Compares the state of the replica in the KumoScale Provisioner to its state in the multiple device driver (md). If discrepancies are detected, corrective measures, such as adding a new leg and removing the faulty one, are applied. A replica marked as deleted, either intentionally or because a storage node is down, will be removed from md.
- If a replica was detected missing
 - for a time greater than the value of **maxReplicaDowntime** for the node. and
 - where the volume has less than four replicas (the maximum).

KumoScale will initiate a process where a new replica will be allocated in the most appropriate location, according to the volume’s storage class parameters. The self-healing process will also connect to the new replica and synchronize it. The missing replica will be detected and removed on the next self-healing iteration.

Only a single repair is executed on each self-healing iteration.

Planned Maintenance

KumoScale software enables you to plan around maintenance operations by providing support for adding a removing a replica to a volume. Using the REST API or other KumoScale interfaces, you can add a replica to a volume, remove the replica from the KumoScale device that is shut down, and create it on a device on a different rack. When the maintenance operation completes, it is possible to return the replica to its original location in the same manner. See the [documentation](#) for your orchestration environment for utilities that facilitate this functionality.

Creating and Using PEM Files with KumoScale

REST clients and KumoScale orchestration interfaces require a Privacy Enhanced Mail (PEM) file that contains a public key certificate. This section shows you how to create such a PEM file and use it with KumoScale

To create a PEM file with the name keystore.pem:

- Generate a Java™ Key Store (JKS) file to use with KumoScale. You can do this using Linux keytool:

```
keytool -genkeypair -alias <key pair entry name> -keyalg RSA -keysize 2048 -keystore <Name-for-JKS-File>.jks -dname CN=<your domain name> -storepass <your password>
```

- Export the single public key certificate out of the JKS file:

```
keytool -importkeystore -srckeystore <Name-for-JKS-File>.jks -destkeystore <Name-for-JKS-File>.p12 -srcstoretype jks -deststoretype pkcs12
```

3. Convert the certificate into PEM format

```
openssl pkcs12 -nodes -in <Name-for-JKS-File>.p12 -out <Name-for-JKS-File>.pem
```

Follow the instructions for your orchestration framework (Ansible, Cluster Manager CLI, CSI, or OpenStack) to replace the PEM file.

Replacing the KumoScale Provisioner Key Store

To replace the KumoScale Provisioner key store where it is installed as a CR:

1. Generate a Java Key Store (JKS) file to use with KumoScale. You can do this using Linux keytool:

```
keytool -genkeypair -alias <key pair entry name> -keyalg RSA -keysize 2048 -keystore <Name-for-JKS-File>.jks -dname CN=<your domain name> -storepass <your password>
```

2. Copy the JKS file above to the KumoScale Provisioner platform with the same permissions as the existing JKS file.

3. Confirm there is a Trusted Certificate entry in addition to the Key Pair entry with:

```
keytool -importcert -alias [trusted certificate entry name] -file [cert file] -keystore [truststore jks file]
```

4. Create a secret file from the JKS file with:

```
kubectl create secret generic ks-jks --from-file=<Name-for-JKS-File> --dry-run -o yaml > ks-jks-secret.yaml
```

5. Edit **ks-jks-secret.yaml** with the following parameter values:

- Under **data**:
 - Set the key for the secret file content to **jksFile**
 - Add a key to **jksPassword** - the value should be the JKS key password encoded to base 64
 - Add a key **jksKeyAlias** - the value should be the jks key alias encoded to base 64
- Under metadata:
 - Set **namespace** to **kumo-services**

6. Create the secret with:

```
kubectl create -f ks-jks-secret.yaml
```

7. Edit and save the Provisioner cr yaml with the new secret:

```
sslSecret = <name of ks-jks-secret>
```

8. Use the Provisioner CR yaml to delete and recreate the Provisioner.

Modifying Secrets

If you need to update a secret for any reason, use **kubectl replace**. For example:

```
kubectl replace -f provisioner-secret-cr.yaml
```

If you use **apply**, sensitive data will be shown when using **kubectl get secret -o yaml**.

Replacing the License

As explained in *KumoScale License* section of [Before you Begin Using KumoScale](#), you will need to replace the KumoScale license if you are moving from a trial to a production. To do this:

1. Edit the license CR file, `kioxia.com_v1_license_cr.yaml`. This is typically located in `Kumoscale Operator/deploy/crds/` and replace the value of `license` with the new license key provided by KIOXIA.
2. Update the license with:

```
kubectl apply -f deploy/crds/kumoscale.kioxia.com_v1_license_cr.yaml
```

Initiator NQN Identifier

KumoScale software supports Access Control Lists (ACL). When the initiator discovers or logs into a target, the initiator must provide its NQN identifier. The default NQN is read from the `/etc/nvme/hostnqn` file located on the initiator server, unless otherwise specified.

Users can either use the script provided by KIOXIA or manually perform the following steps, as shown.

To create the default host NQN identifier for an initiator, use the command sequence below:

1. Create a new folder titled **/etc/nvme/** on the initiator.
2. Use the **nvme** command to create and populate the file `hostnqn` with the initiator's NQN:

```
nvme gen-hostnqn>>/etc/nvme/hostnqn
```

3. The NQN can be viewed and used by the command:

```
cat /etc/nvme/hostnqn
```

Use this NQN identifier to set the initiator of a target within KumoScale software.

Note: An initiator’s name (NQN) cannot be modified once it is set.

KumoScale Field Types

The following table describes input field types for KumoScale parameters.

Field Type	Specification
Name	Up to 16 characters: Alphanumeric, ‘_’ and ‘-’ and must begin with a letter
Portal port	1024-65535

Rebooting a Storage Node

Storage nodes may be rebooted via the REST API reboot command. See the REST API guide for details on how to successfully reboot nodes.

Restore Factory Default Settings when using Appliance Mode

CAUTION: All KumoScale software configurations are lost and data becomes unavailable after this process.

Note: Administrators must re-install licenses if they simultaneously reset to factory defaults for all KumoScale storage nodes

When using Appliance mode, all user configurations and logs may be reset to factory settings using the REST command *Reset Factory Settings*. The data on the SSDs will become inaccessible due to the loss of the volume mapping and initiator configuration.

It is recommended that KumoScale software be removed from the KumoScale Provisioner service prior to the restore operation. If it is not, administrators will have to remove the KumoScale software storage node (after it was restored and configured) and add it again.

Generating Log Files

Administrators may generate and collect KumoScale log files remotely via the REST API.

Currently Running Tasks

Administrators may read the status and properties of user-triggered tasks in KumoScale software using the KumoScale REST API **Get Tasks** command. Ongoing tasks are aborted using the **Remove Task** command.

Locating an SSD

You can locate an SSD by setting a blinking LED on an SSD slot with the REST API **Locate SSD** command.

Support for Multipathing

KumoScale 3.21 and above support NVMe multipathing if it is enabled in the kernel of the initiator (host). To ensure NVMe multipathing is enabled:

- Determine whether NVMe multipathing is enabled in the kernel. For
 - KumoScale 3.22, use the Cluster Manager CLI **host-show** or REST API **List Hosts** commands to identify whether multipathing is enabled (**multipath=Yes**) or disabled (**multipath=No**).
 - KumoScale 3.21, use the Linux command:

```
# cat /sys/module/nvme_core/parameters/multipath
```

One of the following values is returned:
N, if native NVMe multipathing is **disabled**,
Y, if native NVMe multipathing is **enabled**.
By default it is disabled in CentOS 8 and RHEL 8.

- To enable NVMe multipathing, follow the instructions at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_storage_devices/enabling-multipathing-on-nvme-devices_managing-storage-devices#proc_enabling-dm-multipath-on-nvme-devices_enabling-multipathing-on-nvme-devices.

If you need to enable DM multipathing on NVMe devices , you will need to disable native NVMe multipathing as documented in https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_storage_devices/enabling-multipathing-on-nvme-devices_managing-storage-devices#proc_enabling-dm-multipath-on-nvme-devices_enabling-multipathing-on-nvme-devices.

You should also refer to the instructions in the [documentation for the KumoScale interface](#) you are using (Ansible, CSI, OpenStack.)

[1] 'mdraid' is a Linux OS component that controls storage devices. It is referred to as Linux software RAID as it makes RAID use possible without a hardware RAID controller.
