Veritas InfoScale™ 7.4.3 Support for Kubernetes - Linux

Last updated: 2020-11-13

VERITAS

The truth in information.

Legal Notice

This product may contain third-party software for which Veritas is required to provide attribution to the third-party ("Third-Party Programs"). Some of the Third-Party Programs are available under open source or free software licenses. The License Agreement accompanying the Software does not alter any rights or obligations you may have under those open source or free software licenses. Refer to the third-party legal notices document accompanying this Veritas product or available at: https://www.veritas.com/about/legal/license-agreements

Veritas Technologies LLC

2625 Augustine Drive

Santa Clara, CA 95054

http://www.veritas.com

Technical Support

Technical Support maintains support centers globally. All support services will be delivered in accordance with your support agreement and the then-current enterprise technical support policies. For information about our support offerings and how to contact Technical Support, visit our website:

https://www.veritas.com/support

You can manage your Veritas account information at the following URL:

https://my.veritas.com

If you have questions regarding an existing support agreement, please email the support agreement administration team for your region as follows:

Worldwide (except Japan)	CustomerCare@veritas.com
Japan	CustomerCare_Japan@veritas.com

Documentation

Make sure that you have the current version of the documentation. Each document displays the date of the last update on page 2. The latest documentation is available on the Veritas website:

https://sort.veritas.com/documents

Documentation feedback

Your feedback is important to us. Suggest improvements or report errors or omissions to the documentation. Include the document title, document version, chapter title, and section title of the text on which you are reporting. Send feedback to:

infoscaledocs@veritas.com

You can also see documentation information or ask a question on the Veritas community site:

http://www.veritas.com/community/

Veritas Services and Operations Readiness Tools (SORT)

Veritas Services and Operations Readiness Tools (SORT) is a website that provides information and tools to automate and simplify certain time-consuming administrative tasks. Depending on the product, SORT helps you prepare for installations and upgrades, identify risks in your datacenters, and improve operational efficiency. To see what services and tools SORT provides for your product, see the data sheet:

https://sort.veritas.com/data/support/SORT_Data_Sheet.pdf

Contents

Section 1: Overview

Introduction to InfoScale support for Kubernetes

Veritas InfoScale for Containers is a software-defined storage management solution for container-based architectures. The InfoScale architecture is designed for optimal application performance while providing the required resiliency and data protection that is necessary for enterprise applications.

You can deploy and manage containers that host applications by leveraging the storage management and high availability (HA) features of InfoScale Enterprise. The Veritas File System (VxFS) and Veritas Volume Manager (VxVM) components of InfoScale Enterprise provide storage management capabilities to manage storage for containers.

As the container technologies continues to evolve, InfoScale Enterprise supports Kubernetes to orchestrate and schedule containers in InfoScale clusters. While containers virtualize the applications from the underlying virtual or physical infrastructure, Kubernetes orchestrates, and scales containerized applications in real time.

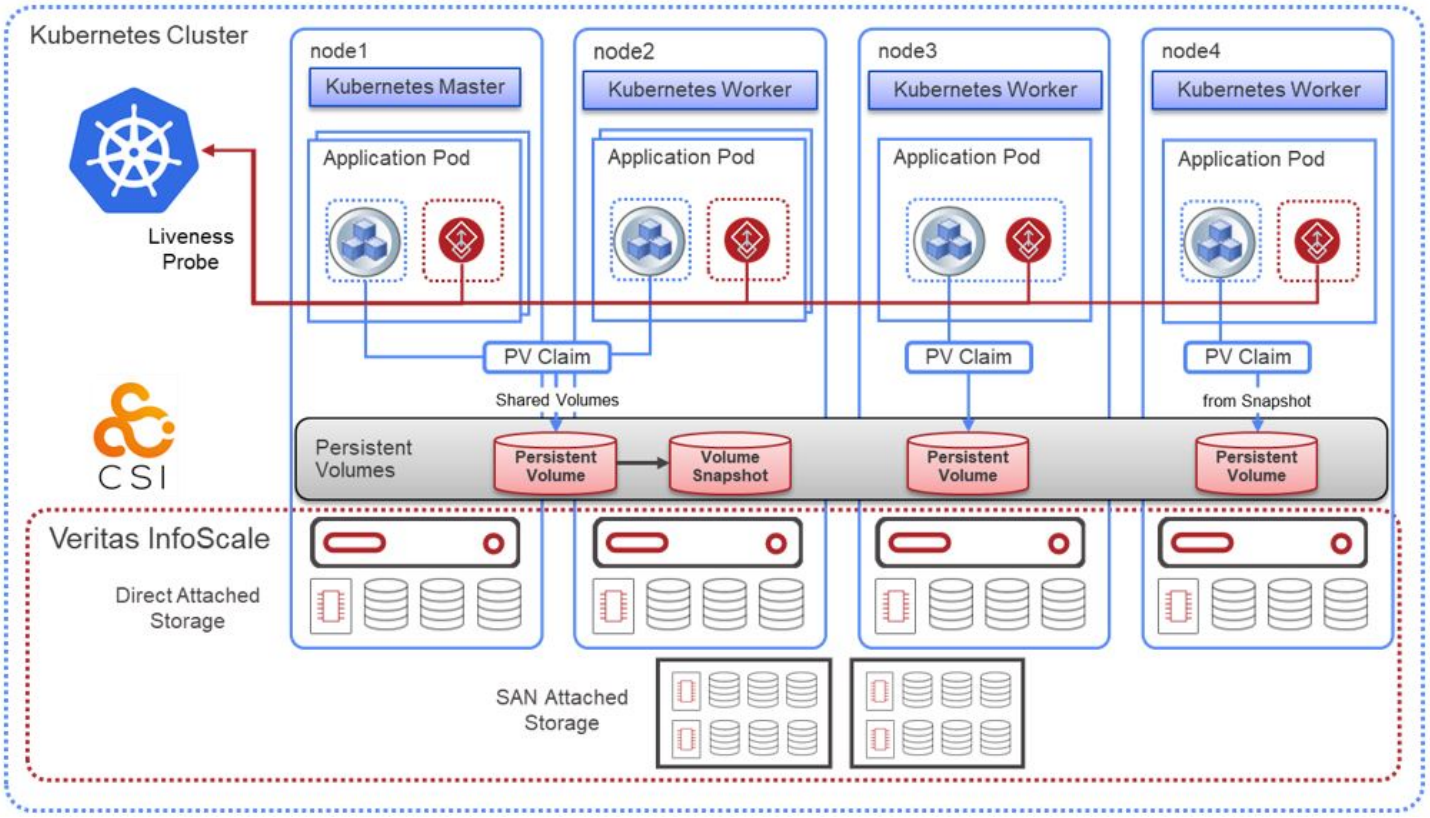
InfoScale provides the following advanced functionality for containers and Kubernetes that is needed by enterprise applications:

- **Application availability** – HA of data is one of the prime considerations when planning for business continuity in case of disasters. InfoScale manages HA and recovery automation for your applications by monitoring critical application processes and resources. InfoScale provides persistent storage for stateful container applications. InfoScale fencing and arbitration prevent data corruption and speed up recovery. The use of VCS agents and persistent volumes ensure that there is no loss of data when containers fail; and data is always available no matter where the containers are scheduled in the container ecosystem.
- **Advanced storage management** – The InfoScale Container Storage Interface (CSI) plugin provides high-performance shared storage for the Kubernetes clusters using the fast storage directly attached to the Kubernetes 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. InfoScale also supports Velero, a third-party application, to provide snapshot lifecycle management.
- **Application migration** – InfoScale supports moving non-containerized applications into a container environment by copying the application data storage volumes onto the Kubernetes cluster nodes. The InfoScale CSI plugin then presents the same data to the applications that are running in the container.

The key benefits to using InfoScale with Kubernetes are:

- High-performance parallel storage that provides better performance and reliability than NFS
- Optimized resource utilization with the ability to use either existing SAN storage or the InfoScale advanced FSS option that provides better performance than SAN at a reduced cost
- Migration of data from legacy to containerized MySQL applications
- Support for in-depth application and infrastructure monitoring with the VCS application HA agents that run inside the containers or inside the sidecar container
- Integrated I/O fencing and arbitration to protect against data corruption and to provide fast recovery in the event of a failure
- Snapshot copies of the production data for analytics and disaster recovery
- Volume cloning to address storage disk and node failures
- Support for statefulset applications like MySQL

The following graphic illustrates a typical InfoScale Enterprise configuration in a supported Kubernetes environment:



A typical InfoScale Enterprise configuration in a Kubernetes environment comprises the following components:

Kubernetes: Kubernetes is responsible for the deployment and the management of infrastructure. A Kubernetes cluster can have more than one master and multiple nodes to ensure that there is no single-point-of-failure in the setup. Kubernetes provides a framework to run distributed systems resiliently, and it takes care of:

- Self-healing and container restarts through user-defined health check, probes, and monitoring
- Easy deployments with yaml files, and so on

InfoScale: InfoScale Enterprise provides application availability and data resiliency across physical and virtual infrastructures. Along with the other important features, InfoScale uses the following components for monitoring and storage provisioning:

- **InfoScale Availability (formerly Cluster Server—VCS) for Containers**: The VCS component monitors and orchestrates HA for mission-critical enterprise applications. It does so by monitoring the application and its resources, orchestrating a failover when a fault is detected, and then performing a failback after the fault is addressed. You can deploy VCS for containers running on Kubernetes either as a sidecar container, as part of the pod that contains the application and other containers, or within the application container itself.
- **InfoScale Storage**: The Storage components together provide a software-defined storage solution. They provide highly available persistent storage that conform to CSI specifications for enterprise applications by using high-performance parallel storage access on shared storage (SAN) or in Flexible Storage Sharing (FSS) environments.

InfoScale CSI plugin: CSI is a standardized mechanism for Container Orchestration Systems (COSs) to expose arbitrary storage systems to their containerized workloads. The InfoScale CSI plugin is used to provide persistent InfoScale storage to container satefulsets and supports creation of storage classes for HA, performance, capacity, and data security. It also supports online expansion of capacity as well as the usage of snapshots and clones.

Section 2: Planning and preparation

Licensing Veritas InfoScale

About Veritas InfoScale product licensing

You must obtain a license to install and use Veritas InfoScale products. You can choose one of the following licensing methods when you install a product:

- Install product with a permanent license

  When you purchase a Veritas InfoScale product, you receive a License Key certificate. The certificate specifies the products and the number of product licenses purchased.

  See Registering Veritas InfoScale using permanent license key file.

- Install product without a permanent license key (keyless licensing)

  Installation without a license does not eliminate the need to obtain a license. The administrator and company representatives must ensure that a server or cluster is entitled to the license level for the products installed. Veritas reserves the right to ensure entitlement and compliance through auditing.

  See Registering Veritas InfoScale using keyless license.

- Veritas collects licensing and platform related information from InfoScale products as part of the Veritas Product Improvement Program. The information collected helps identify how customers deploy and use the product, and enables Veritas to manage customer licenses more efficiently. See About telemetry data collection in InfoScale.

  Visit the Veritas licensing Support website, for more information about the licensing process.

  www.veritas.com/licensing/process

About telemetry data collection in InfoScale

The Veritas Telemetry Collector is used to collect licensing and platform related information from InfoScale products as part of the Veritas Product Improvement Program. The Veritas Telemetry Collector sends this information to an edge server.

The information collected helps identify how customers deploy and use the product, and enables Veritas to manage customer licenses more efficiently. The edge server does not collect any private information and only uses information specific to product, licensing, and platform (which includes operating system and server hardware).

Category	Information attributes
Product	• Telemetry data version • Cluster ID • Product version • Time stamp
Licensing	• Product ID • Serial number • Serial ID • License meter • Fulfillment ID • Platform

Category	Information attributes
	• Version • SKU type • VXKEYLESS • License type • SKU
Operating system	• Platform name • Version • TL number • Kernel/SRU
Server hardware	• Architecture • CPU op-mode(s) • CPU(s) • Core(s) per socket • Thread(s) per core • Socket(s) • Vendor ID • CPU model name • CPU frequency • Hypervisor vendor • Memory

By default, the Veritas Telemetry Collector will collect telemetry data every Tuesday at 1:00 A.M. as per the local system time. The time and interval of data collection can be customized by the user if required.

You can configure the Veritas Telemetry Collector while installing or upgrading the product, See Installing Veritas InfoScale using the product installer. You can also manage the Veritas Telemetry Collector on each of your servers by using the /opt/VRTSvlic/tele/bin/TelemetryCollector command.

Configure the firewall policy such that the ports required for telemetry data collection are not blocked. Refer to your respective firewall or OS vendor documents for the required configuration.

Note: Ensure that you reboot the server after uninstalling the product to ensure that all services related to the Veritas Telemetry Collector are stopped successfully.

Licensing notes

Review the following licensing notes before you install or upgrade the product.

- If you use a keyless license option, you must configure Veritas InfoScale Operations Manager within two months of product installation and add the node as a managed host to the Veritas InfoScale Operations Manager Management Server. Failing this, a warning message for non-compliance is displayed periodically.

  For more details, refer to Veritas InfoScale Operations Manager product documentation.
- Note the following limitation in case of InfoScale Availability and InfoScale Storage co-existence:
- If Keyless licensing type is selected during the product installation, checks performed to monitor the number of days of product installation are based on the InfoScale Storage component. As a result, if you do not enter a valid license key file or do not add the host as a managed host within 60 days of InfoScale Storage installation, a non-compliance error is logged every 4 hrs in the Event Viewer.
- The text-based license keys that are used in 7.3.1 and earlier versions are not supported when upgrading to later versions. If your current product is installed using a permanent license key and you do not have a permanent license key file for the newer InfoScale version, you can temporarily upgrade using the keyless licensing. Then you must procure a permanent license key file from the Veritas license certificate and portal within 60 days, and upgrade using the permanent license key file to continue using the product.
- The license key file must be present on the same node where you are trying to install the product.

  Note: The license key file must not be saved in the root directory (/) or the default license directory on the local host (/etc/vx/licesnes/lic). You can save the license key file inside any other directory on the local host.
- You can manage the license keys using the vxlicinstupgrade utility. See Managing InfoScale licenses.
- Before upgrading the product, review the licensing details and back up the older license key. If the upgrade fails for some reason, you can temporarily revert to the older product using the older license key to avoid any application downtime.
- You can use the license assigned for higher Stock Keeping Units (SKU) to install the lower SKUs.
- For example, if you have procured a license that is assigned for InfoScale Enterprise, you can use the license for installing any of the following products:
  - InfoScale Storage
  - InfoScale Availability

  The following table provides details about the license SKUs and the corresponding products that can be installed with Kubernetes:

License SKU procured	**Products that can be installed**		
	InfoScale Storage	InfoScale Availability	InfoScale Enterprise
InfoScale Storage	✓	X	X
InfoScale Availability	X	✓	X
InfoScale Enterprise	✓	✓	✓

  Note: At any given point in time you can install only one product.

Registering Veritas InfoScale using permanent license key file

If license key files are required while registering Veritas InfoScale using a permanent license key file. Ensure that the license key file is downloaded on the local host, where you want to install or upgrade the product.

Note: The license key file must not be saved in the root directory (/) or the default license directory on the local host (/etc/vx/licesnes/lic). You can save the license key file inside any other directory on the local host.

You can register your permanent license key file in the following ways:

Method	Description

Using the installer	You can register your InfoScale product using a permanent license key file during the installation process.
	• Run the following command:
	`./installer`
	• During the installation, the following interactive message appears:
	`1) Enter a valid license key(key file path needed)`
	`2) Enable keyless licensing and complete system licensing later`
	`How would you like to license the systems? [1-2,q] (2)`
	• Enter 1 to register the license key.
	• Provide the absolute path of the `.slf` license key file saved on the current node.
	Example:
	/`downloads`/InfoScale_keys/XYZ.slf
	Alternatively, you can register your InfoScale product using the installer menu.
	• Run the following command:
	`./installer`
	• Select the L) License a Product option in the installer menu.
	• Then proceed to provide the licensing details as prompted.
	To install InfoScale using the installer:
	See Installing Veritas InfoScale using the product installer.

Method	Description

Manual	If you are performing a fresh installation, run the following commands on each node:
	`# cd /opt/VRTS/bin`
	`# ./vxlicinstupgrade -k <key file path>`
	or
	`# ./vxlicinst -k <key file path>`
	then,
	`# vxdctl license init`
	Note: It is recommended to use the `vxlicinstupgrade` utility to manage licenses. The `vxlicinst` utility is expected to be deprecated in near future.
	If you are performing an upgrade, run the following commands on each node:
	`# cd /opt/VRTS/bin`
	`# ./vxlicinstupgrade -k <key file path>`
	For more information, see Managing InfoScale licenses.

Even though other products are included on the enclosed software discs, you can only use the Veritas InfoScale software products for which you have purchased a license.

Registering Veritas InfoScale using keyless license

You can enable keyless licensing for your product in the following ways:

Method	Description

Using the installer	You can enable keyless licensing for InfoScale during the installation process.
	• Run the following command:
	`./installer`
	• During the installation, the following interactive message appears:
	`1) Enter a valid license key(key file path needed)`
	`2) Enable keyless licensing and complete system licensing later`
	`How would you like to license the systems? [1-2,q] (2)`
	• Enter 2 to enable keyless licensing.
	Alternatively, you can enable keyless licensing for your InfoScale product using the installer menu.
	• Run the following command:
	`./installer`
	• Select the L) License a Product option in the installer menu.
	• Then proceed to provide the licensing details as prompted.
	To install InfoScale using the installer:
	See Installing Veritas InfoScale using the product installer.

Manual	If you are performing a fresh installation, run the following commands on each node:
	1. Change your current working directory:
	`# export PATH=$PATH:/opt/VRTSvlic/bin`
	2. View the keyless product code for the product you want to install:
	`# vxkeyless displayall`
	3. Enter the product code in the exact format as displayed in the previous step:
	`# vxkeyless set <product code>`
	Example:
	`# vxkeyless set ENTERPRISE`
	For more information: See Managing InfoScale licenses.

Warning: Within 60 days of choosing this option, you must install a valid license key file corresponding to the license level entitled, or continue with keyless licensing by managing the systems with Veritas InfoScale Operation Manager. If you fail to comply with the above terms, continuing to use the Veritas InfoScale product is a violation of your End User License Agreement, and results in warning messages.

For more information about keyless licensing, see the following URL:

http://www.veritas.com/community/blogs/introducing-keyless-featureenablement-storage-foundation-ha-51

For more information to use keyless licensing and to download the Veritas InfoScale Operation Manager, see the following URL:

www.veritas.com/product/storage-management/infoscale-operations-manager

Managing InfoScale licenses

After you have installed a Veritas InfoScale product, you may need to manage the product license, for example, to switch from a keyless to a permanent license type.

You can manage your licenses by using the `vxlicinstupgrade` or `vxkeyless` utilities which are located in the product installation directory.

Method	Description

Using the `vxlicinstupgrade`	To add or update a permanent license, run the following commands:
	`# cd /opt/VRTS/bin`
	`# ./vxlicinstupgrade -k <key file path>`
	Where, the <key file path> is the absolute path of the .slf license key file saved on the current node.
	Example:
	`/downloads/InfoScale_keys/XYZ.slf`
	For more information on vxlicinstupgrade utility, see About the vxlicinstupgrade utility.
	For more information on permanent licensing, see Registering Veritas InfoScale using permanent license key file.

Using the `vxkeyless`	To add or update a keyless license, perform the following steps:
	1. Change your current working directory:
	`# export PATH=$PATH:/opt/VRTSvlic/bin`
	2. View the keyless product code for the product you want to install:
	`# vxkeyless displayall`
	3. Enter the product code in the exact format as displayed in the previous step:
	`# vxkeyless set <keyless license text-string>`
	Example:
	`# vxkeyless set ENTERPRISE`
	For more information on keyless licensing, see Registering Veritas InfoScale using keyless license.

About the vxlicinstupgrade utility

The `vxlicinstupgrade` utility enables you to perform the following tasks:

- Upgrade to another Veritas InfoScale license
- Update a keyless license to a permanent license
- Manage co-existence of multiple licenses

  On executing the `vxlicinstupgrade` utility, the following checks are done:
- If the current license is keyless or permanent and if the user is trying to install the keyless or permanent license of the same product.

  Example: If the 7.4.2 Foundation Keyless license key is already installed on a system and the user tries to install another 7.4.2 Foundation Keyless license key, then `vxlicinstupgrade` utility shows an error message:

  `vxlicinstupgrade` WARNING: The input License key and Installed key are same.
- If the current key is keyless and the newly entered license key file is a permanent license of the same product

  Example: If the 7.4.2 Foundation Keyless license key is already installed on a system and the user tries to install 7.4.2 Foundation permanent license key file, then the `vxlicinstupgrade` utility installs the new license at /etc/vx/licenses/lic and the 7.4.2 Foundation Keyless key is deleted.
- The `vxlicinstupgrade` utility in Veritas InfoScale does not support managing the text-based license keys used in versions before 7.4.
- If the current key is of a lower version and the user tries to install a higher version license key.

  Example: If 7.0 Storage license key is already installed on a system and the user tries to install 7.4.2 Storage license key file, then the `vxlicinstupgrade` utility installs the new license at /etc/vx/licenses/lic and the 7.0 Storage key is deleted.

  Note: When registering license key files manually during upgrade, you have to use the `vxlicinstupgrade` command. When registering keys using the installer script, the same procedures are performed automatically.

Generating license report with vxlicrep command

The `vxlicrep` command generates a report of the product licenses in use on your system.

To display a license report:

- Enter the `# vxlicrep` command without any options to display the report of all the product licenses on your system, or
- Enter the `# vxlicrep` command with any of the following options to display the type of report required:

  -g default report

  -k <key> print report for input key

  -v print version

  -h display this help

System requirements

Disk space requirements

Table 1 lists the minimum disk space requirements for RHEL and supported RHEL-compatible distributions for each product when the /opt, /root, /var, and /bin directories are created on the same disk.

Table 1: Disk space requirements for RHEL and supported RHEL-compatible distributions

Product name	RHEL 7 (MB)	RHEL 8 (MB)
Veritas InfoScale Availability	2329	1810
Veritas InfoScale Storage	3852	3305
Veritas InfoScale Enterprise	3959	3407

Table 2 lists the minimum disk space requirements for SLES each product when the /opt, /root, /var, and /bin directories are created on the same disk.

Table 2: Disk space requirements

Product name	SLES 12 (MB)	SLES 15 (MB)
Veritas InfoScale Availability	2391	2216
Veritas InfoScale Storage	4341	3583
Veritas InfoScale Enterprise	4463	3683

Hardware requirements

This section lists the hardware requirements for Veritas InfoScale.

Table 3 lists the hardware requirements for each component in Veritas InfoScale.

Table 3: Hardware requirements for components in Veritas InfoScale

Component	Requirement
Storage Foundation for High Availability (SFHA)	See SFHA hardware requirements
Storage Foundation Cluster File System (SFCFS) and Storage Foundation Cluster File System for High Availability (SFCFSHA)	See SFCFS and SFCFSHA hardware requirements
Cluster Server (VCS)	See VCS hardware requirements

For additional information, see the hardware compatibility list (HCL) at:

https://www.veritas.com/support/en_US/article.000126344

SFHA hardware requirements

Table 4 lists the hardware requirements for SF and SFHA.

Table 4: SF and SFHA hardware requirements

Item	Requirement
Memory	Each system requires at least 1 GB.

SFCFS and SFCFSHA hardware requirements

Table 5 lists the hardware requirements for SFCFSHA.

Table 5: Hardware requirements for SFCFSHA

Requirement	Description
Memory (Operating System)	2 GB of memory.
CPU	A minimum of 2 CPUs.
Node	All nodes in a Cluster File System must have the same operating system version.
Shared storage	Shared storage can be one or more shared disks, or a disk array connected either directly to the nodes of the cluster or through a Fibre Channel Switch. Nodes can also have non-shared or local devices on a local I/O channel. It is advisable to have `/`, `/usr`, `/var` and other system partitions on local devices. In a Flexible Storage Sharing (FSS) environment, shared storage may not be required.
Fibre Channel or iSCSI storage	Each node in the cluster must have a Fibre Channel I/O channel or iSCSI storage to access shared storage devices. The primary component of the Fibre Channel fabric is the Fibre Channel switch.
Cluster platforms	There are several hardware platforms that can function as nodes in a Veritas InfoScale cluster. See the Veritas InfoScale 7.4.3 Release Notes. For a cluster to work correctly, all nodes must have the same time. If you are not running the Network Time Protocol (NTP) daemon, make sure the time on all the systems comprising your cluster is synchronized.
SAS or FCoE	Each node in the cluster must have an SAS or FCoE I/O channel to access shared storage devices. The primary components of the SAS or Fibre Channel over Ethernet (FCoE) fabric are the switches and HBAs.

VCS hardware requirements

Table 6 lists the hardware requirements for a VCS cluster.

Table 6: Hardware requirements for a VCS cluster

Item	Description
DVD drive	One drive in a system that can communicate to all the nodes in the cluster.
Disks	Typical configurations require that the applications are configured to use shared disks/storage to enable migration of applications between systems in the cluster. The SFHA I/O fencing feature requires that all data and coordinator disks support SCSI-3 Persistent Reservations (PR). **Note**: SFHA also supports non-SCSI3 server-based fencing configuration in virtual environments that do not support SCSI-3 PR-compliant storage.
Network Interface Cards (NICs)	In addition to the built-in public NIC, VCS requires at least one more NIC per system. Veritas recommends two additional NICs. You can also configure aggregated interfaces. Veritas recommends that you turn off the spanning tree on the LLT switches, and set port-fast on.
Fibre Channel or SCSI host bus adapters	Typical VCS configuration requires at least one SCSI or Fibre Channel Host Bus Adapter per system for shared data disks.
RAM	Each VCS node requires at least 1024 megabytes.

Supported operating systems and database versions

For information on supported operating systems and database versions for various components of Veritas InfoScale, see the Veritas InfoScale Release Notes.

Number of nodes supported

Veritas InfoScale recommends cluster configurations up to 8 nodes.

SFHA, SFCFSHA, SF Oracle RAC: Flexible Storage Sharing (FSS) only supports cluster configurations with up to 64 nodes.

SFHA, SFCFSHA: SmartIO writeback caching only supports cluster configurations with up to 2 nodes.

Supported application

Along with the generic applications, InfoScale supports Nginix and MySQL in a Kubernetes environment.

Preparing to install

Mounting the ISO image

An ISO file is a disc image that must be mounted to a virtual drive for use. You must have superuser (root) privileges to mount the Veritas InfoScale ISO image.

To mount the ISO image

1. Log in as superuser on a system where you want to install Veritas InfoScale.
2. Mount the image:
   ```
   # mount -o loop <ISO_image_path> /mnt
   ```

Setting up ssh or rsh for inter-system communications

The installer uses passwordless Secure Shell (ssh) or Remote Shell (rsh) communications among systems. During an installation, you choose the communication method that you want to use. Or, you can run the `installer -comsetup` command to set up ssh or rsh explicitly. When the installation process completes, the installer asks you if you want to remove the password-less connection. If installation terminated abruptly, use the installation script's `-comcleanup` option to remove the ssh or rsh configuration from the systems.

In most installation, configuration, upgrade (where necessary), and uninstallation scenarios, the installer configures ssh or rsh on the target systems. When you perform installation using a response file, you need to set up ssh or rsh manually, or use the `installer -comsetup` option to set up an ssh or rsh configuration from the systems.

Obtaining installer patches

You can access public installer patches automatically or manually on the Veritas Services and Operations Readiness Tools (SORT) website's Patch Finder page at:

https://sort.veritas.com/patch/finder

To download installer patches automatically

If you are running Veritas InfoScale version 7.0 or later, and your system has Internet access, the installer automatically imports any needed installer patch, and begins using it.

Automatically downloading installer patches requires the installer to make outbound networking calls. You can also disable external network connection attempts.

See Disabling external network connection attempts

If your system does not have Internet access, you can download installer patches manually.

To download installer patches manually

1. Go to the Veritas Services and Operations Readiness Tools (SORT) website's Patch Finder page, and save the most current patch on your local system.
2. Navigate to the directory where you want to unzip the file you downloaded in step 1.
3. Unzip the patch tar file. For example, run the following command:
   ```
   # gunzip cpi-7.4.3P1-patches.tar.gz
   ```
4. Untar the file. For example, enter the following:
   ```
   # tar -xvf cpi-7.4.3P1-patches.tar
   patches/
   patches/CPI7.4.3P1.pl
   ```

```
README
```

5. Navigate to the installation media or to the installation directory.
6. To start using the patch, run the installer command with the -require option.

   For example, enter the following:

   ```
   # ./installer -require /target_directory/patches/CPI7.4.3P1.pl
   ```

Disabling external network connection attempts

When you execute the installer command, the installer attempts to make an outbound networking call to get information about release updates and installer patches. If you know your systems are behind a firewall, or do not want the installer to make outbound networking calls, you can disable external network connection attempts by the installer.

To disable external network connection attempts

- Disable inter-process communication (IPC).

  To disable IPC, run the installer with the -noipc option.

  For example, to disable IPC for system1 (sys1) and system2 (sys2) enter the following:

  ```
  # ./installer -noipc sys1 sys2
  ```

Verifying the systems before installation

Use any of the following options to verify your systems before installation:

- **Option 1**: Run Veritas Services and Operations Readiness Tools (SORT).

  For information on downloading and running SORT:

  https://sort.veritas.com

  Note: You can generate a pre-installation checklist to determine the pre-installation requirements: Go to the SORT installation checklist tool. From the drop-down lists, select the information for the Veritas InfoScale product you want to install, and click **Generate Checklist**.
- **Option 2**: Run the installer with the "-precheck" option as follows:

  Navigate to the directory that contains the installation program.

  Start the preinstallation check:

  ```
  # ./installer -precheck sys1 sys2
  ```

  where sys1, sys2 are the names of the nodes in the cluster.

  The program proceeds in a non-interactive mode, examining the systems for licenses, RPMs, disk space, and system-to-system communications. The program displays the results of the check and saves them in a log file. The location of the log file is displayed at the end of the precheck process.

Setting up the private network

This topic applies to VCS, SFHA, SFCFS, SFCFSHA, SF Oracle RAC, and SF Sybase CE.

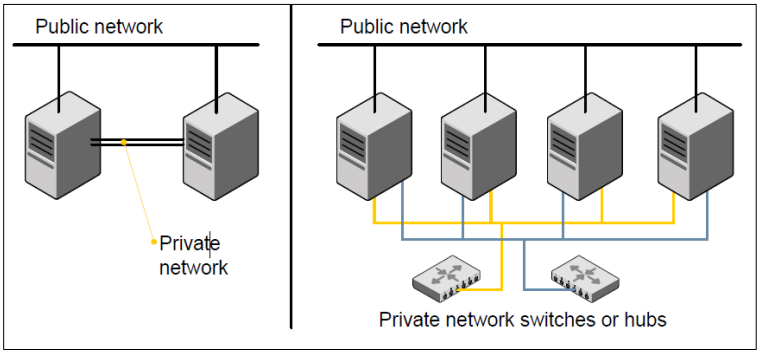
VCS requires you to set up a private network between the systems that form a cluster. You can use either NICs or aggregated interfaces to set up private network.

You can use network switches instead of hubs.

Refer to the Cluster Server Administrator's Guide to review VCS performance considerations.

Figure 1 shows two private networks for use with VCS.

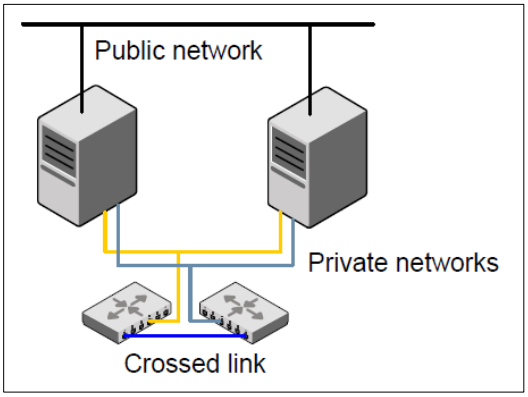
Figure 1: Private network setups: two-node and four-node clusters



You need to configure at least two independent networks between the cluster nodes with a network switch for each network. You can also interconnect multiple layer 2 switches for advanced failure protection. Such connections for LLT are called cross-links.

Figure 2 shows a private network configuration with crossed links between the network switches.

Figure 2: Private network setup with crossed links



Veritas recommends one of the following two configurations:

- Use at least two private interconnect links and one public link. The public link can be a low priority link for LLT. The private interconnect link is used to share cluster status across all the systems, which is important for membership arbitration and high availability. The public low priority link is used only for heartbeat communication between the systems.
- If your hardware environment allows use of only two links, use one private interconnect link and one public low priority link. If you decide to set up only two links (one private and one low priority link), then the cluster must be configured to use I/O fencing, either disk-based or server-based fencing configuration. With only two links, if one system goes down, I/O fencing ensures that other system can take over the service groups and shared file systems from the failed node.

To set up the private network

1. Install the required network interface cards (NICs).

   Create aggregated interfaces if you want to use these to set up private network.
2. Connect the Veritas InfoScale private NICs on each system.
3. Use crossover Ethernet cables, switches, or independent hubs for each Veritas InfoScale communication network. Note that the crossover Ethernet cables are supported only on two systems.

   Ensure that you meet the following requirements:
   - The power to the switches or hubs must come from separate sources.

- On each system, you must use two independent network cards to provide redundancy.
- If a network interface is part of an aggregated interface, you must not configure the network interface under LLT. However, you can configure the aggregated interface under LLT.
- When you configure Ethernet switches for LLT private interconnect, disable the spanning tree algorithm on the ports used for the interconnect.

During the process of setting up heartbeat connections, consider a case where a failure removes all communications between the systems.

Note that a chance for data corruption exists under the following conditions:

- The systems still run, and
- The systems can access the shared storage.

4. Test the network connections. Temporarily assign network addresses and use telnet or ping to verify communications.

   LLT uses its own protocol and does not use TCP/IP. So, you must ensure that the private network connections are used only for LLT communication and not for TCP/IP traffic. To verify this requirement, unplumb and unconfigure any temporary IP addresses that are configured on the network interfaces.

   The installer configures the private network in the cluster during configuration. You can also manually configure LLT.

Optimizing LLT media speed settings on private NICs

For optimal LLT communication among the cluster nodes, the interface cards on each node must use the same media speed settings. Also, the settings for the switches or the hubs that are used for the LLT interconnections must match that of the interface cards. Incorrect settings can cause poor network performance or even network failure.

If you use different media speed for the private NICs, Veritas recommends that you configure the NICs with lesser speed as low-priority links to enhance LLT performance.

Guidelines for setting the media speed for LLT interconnects

Review the following guidelines for setting the media speed for LLT interconnects:

- Veritas recommends that you manually set the same media speed setting on each Ethernet card on each node.

  If you use different media speed for the private NICs, Veritas recommends that you configure the NICs with lesser speed as low-priority links to enhance LLT performance.
- If you have hubs or switches for LLT interconnects, then set the hub or switch port to the same setting as used on the cards on each node.

  Details for setting the media speeds for specific devices are outside of the scope of this manual. Consult the device's documentation or the operating system manual for more information.

Guidelines for setting the maximum transmission unit (MTU) for LLT interconnects in Flexible Storage Sharing (FSS) environments

Review the following guidelines for setting the MTU for LLT interconnects in FSS environments:

- Set the maximum transmission unit (MTU) to the highest value (typically 9000) supported by the NICs when LLT (both high priority and low priority links) is configured over Ethernet or UDP. Ensure that the switch is also set to 9000 MTU.

  Note: MTU setting is not required for LLT over RDMA configurations.
- For virtual NICs, all the components—the virtual NIC, the corresponding physical NIC, and the virtual switch—must be set to 9000 MTU.

- If a higher MTU cannot be configured on the public link (because of restrictions on other components such as a public switch), do not configure the public link in LLT. LLT uses the lowest of the MTU that is configured among all high priority and low priority links.

Setting up shared storage

This topic applies to VCS, SFHA, SFCFSHA, SF Oracle RAC, and SF Sybase CE.

The sections describe how to set up the SCSI and the Fibre Channel devices that the cluster systems share.

Setting up shared storage: SCSI

Perform the following steps to set up shared storage.

To set up shared storage

1. Connect the disk to the first cluster system.
2. Power on the disk.
3. Connect a terminator to the other port of the disk.
4. Boot the system. The disk is detected while the system boots.
5. Press CTRL+A to bring up the SCSI BIOS settings for that disk.
   Set the following:
   - Set Host adapter SCSI ID = 7, or to an appropriate value for your configuration.
   - Set Host Adapter BIOS in Advanced Configuration Options to Disabled.
6. Format the shared disk and create required partitions on it.
   Perform the following:
   - Identify your shared disk name. If you have two internal SCSI hard disks, your shared disk is /dev/sdc.
     Identify whether the shared disk is sdc, sdb, and so on.
   - Type the following command:
     `# fdisk /dev/shareddiskname`
     For example, if your shared disk is sdc, type:
     `# fdisk /dev/sdc`
   - Create disk groups and volumes using Volume Manager utilities.
   - To apply a file system on the volumes, type:
     `# mkfs -t fs-type /dev/vx/dsk/disk-group/volume`
     For example, enter the following command:
     `# mkfs -t vxfs /dev/vx/dsk/dg/vol01`
     Where the name of the disk group is dg, the name of the volume is vol01, and the file system type is vxfs.
7. Power off the disk.
8. Remove the terminator from the disk and connect the disk to the other cluster system.
9. Power on the disk.
10. Boot the second system. The system can now detect the disk.
11. Press Ctrl+A to bring up the SCSI BIOS settings for the disk.
    Set the following:
    - Set Host adapter SCSI ID = 6, or to an appropriate value for your configuration. Note that the SCSI ID should be different from the one configured on the first cluster system.
    - Set Host Adapter BIOS in Advanced Configuration Options to Disabled.

12. Verify that you can view the shared disk using the `fdisk` command.

Setting up shared storage: Fibre Channel

Perform the following steps to set up Fibre Channel.

To set up shared storage for Fibre Channel

1. Connect the Fibre Channel disk to a cluster system.
2. Boot the system and change the settings of the Fibre Channel. Perform the following tasks for all QLogic adapters in the system:
   - Press Alt+Q to bring up the QLogic adapter settings menu.
   - Choose Configuration Settings.
   - Click Enter.
   - Choose Advanced Adapter Settings.
   - Click **Enter**.
   - Set the Enable Target Reset option to Yes (the default value).
   - Save the configuration.
   - Reboot the system.
3. Verify that the system detects the Fibre Channel disks properly.
4. Create volumes. Format the shared disk and create required partitions on it and perform the following:
   - Identify your shared disk name. If you have two internal SCSI hard disks, your shared disk is /dev/sdc.
   - Identify whether the shared disk is sdc, sdb, and so on.
   - Type the following command:
     ```
     # fdisk /dev/shareddiskname
     ```
     For example, if your shared disk is sdc, type:
     ```
     # fdisk /dev/sdc
     ```
   - Create disk groups and volumes using Volume Manager utilities.
   - To apply a file system on the volumes, type:
     ```
     # mkfs -t fs-type /dev/vx/rdsk/disk-group/volume
     ```
     For example, enter the following command:
     ```
     # mkfs -t vxfs /dev/vx/rdsk/dg/vol01
     ```
     Where the name of the disk group is dg, the name of the volume is volo1, and the file system type is vxfs.
5. Repeat step 2 and step 3 for all nodes in the clusters that require connections with Fibre Channel.
6. Power off this cluster system.
7. Connect the same disks to the next cluster system.
8. Turn on the power for the second system.
9. Verify that the second system can see the disk names correctly—the disk names should be the same.

Synchronizing time settings on cluster nodes

Make sure that the time settings on all cluster nodes are synchronized. If the nodes are not in sync, timestamps for change (ctime) and modification (mtime) may not be consistent with the sequence in which operations actually happened.

For instructions, see the operating system documentation.

Setting the kernel.hung_task_panic tunable

The topic applies to SFHA, SFCFSHA, and VCS.

By default, in the Linux kernel the kernel.hung_task_panic tunable is enabled and the kernel.hung_task_timeout_secs tunable is set to a default non-zero value.

To ensure that the node does not panic, the kernel.hung_task_panic tunable must be disabled. If kernel.hung_task_panic is enabled, then it causes the kernel to panic when any of the following kernel threads waits for more than the kernel.hung_task_timeout_secs value:

- The vxfenconfig thread in the vxfen configuration path waits for GAB to seed.
- The vxfenswap thread in the online coordinator disks replacement path waits for the snapshot of peer nodes of the new coordinator disks.

To disable the kernel.hung_task_panic tunable:

- Set the kernel.hung_task_panic tunable to zero (0) in the /etc/sysctl.conf file. This step ensures that the change is persistent across node restarts.
- Run the command on each node.

  # sysctl -w kernel.hung_task_panic=0

To verify the kernel.hung_task_panic tunable value, run the following command:

- # sysctl -a | grep hung_task_panic

Section 3: Installation of Veritas InfoScale

Installing InfoScale using the product installer

The product installer is the recommended method to license and install Veritas InfoScale.

Prerequisites:

- Ensure that each Kubernetes cluster node is in READY state where you want to install InfoScale
- Ensure that the Kubernetes master nodes are a part of the InfoScale cluster

To install Veritas InfoScale

1. Load and mount the software disc. If you downloaded the software, navigate to the top level of the download directory and skip the next step.
2. Move to the top-level directory on the disc.
   ```
   # cd /mnt/cdrom
   ```
3. From this directory, type the following command to start the installation on the local system.
   ```
   # ./installer
   ```
4. Press `I` to install and press Enter.
5. The list of available products is displayed. Select the product that you want to install on your system.
   ```
   1)  Veritas InfoScale Foundation
   2)  Veritas InfoScale Availability
   3)  Veritas InfoScale Storage
   4)  Veritas InfoScale Enterprise
   b)  Back to previous menu
   Select a product to install: [1-4,b,q]
   ```
6. The installer asks whether you want to configure the product.

   Enter y to configure the product after installation. If you enter n, the installer quits after the installation is complete.
7. The list of components is displayed. Select the component that you want to configure on your system.

   Note: With Kubernetes, you can only install the VCS, SFHA, and SFCFSHA components.
8. At the prompt, specify whether you accept the terms of the End User License Agreement (EULA).
   ```
   Do you agree with the terms of the End User License Agreement as specified in the
   EULA/en/EULA.pdf file present on media? [y,n,q,?] y
   ```
9. The installer performs the pre-checks. If it is a fresh system, the product is set as the user defined it. If the system already has a different product installed, the product is set as Veritas InfoScale Enterprise with a warning message after pre-check.
   ```
   Veritas InfoScale Availability is installed. Installation of two
   products is not supported, Veritas InfoScale Enterprise will be
   installed to include Veritas InfoScale Storage and Veritas
   InfoScale Availability on all the systems.
   ```
10. Choose the licensing method. Answer the licensing questions and follow the prompts.
    ```
    1)  Enter a valid license key (key file path needed)
    2)  Enable keyless licensing and complete system licensing later
    How would you like to license the systems? [1-2,q]  (2)
    ```
11. Select the component that you want to license and use.
    ```
         1)  Veritas InfoScale Foundation
    ```

```
     2)  Veritas InfoScale Availability
     3)  Veritas InfoScale Storage
     4)  Veritas InfoScale Enterprise
     b)  Back to previous menu
Which product would you like to register? [1-4,b,q] (4)
```

12. An edge server is used to collect telemetry data that will be used to help users monitor their InfoScale licenses more effectively. Enter the host name or IP address of the edge server that you want to use.

```
The Veritas Cloud Receiver (VCR) is a preconfigured, cloud-based
edge server deployed by Veritas. Enter telemetry.veritas.com to
use the Veritas Cloud Receiver as an edge server for your
environment.
Enter the edge server's hostname/ip: [q,?] ?
```

The Veritas Cloud Receiver (VCR) is a preconfigured, cloud-based edge server deployed by Veritas. Enter telemetry.veritas.com to use the Veritas Cloud Receiver as an edge server for your environment. For more information, See About telemetry data collection in InfoScale.

13. The installer starts InfoScale container setup for Kubernetes and asks whether you want to configure container storage in Enabled mode.

```
Do you want to configure container storage in Enabled mode? [y,n,q,?] (n) y
```

Enter y to configure the container setup.

14. As an integrated configuration process, the installer continues to configure the I/O fencing feature. Press Enter to continue.

15. The installer then prompts to configures the VCS component. Press Enter to continue.

On the prompt, Enter the following:

- A unique cluster name for the cluster
- IP address to configure private heartbeat links using LLT over UDP
- IP address to configure low-priority heartbeat links
- A unique cluster ID number between 0-65535

When prompted, enter y to check if the cluster ID is in use by another cluster.

Note: While configuring the heartbeat links, the prompt displays all the existing interface names. Ensure that you select the name of an interface that you want to use for communication within the InfoScale cluster nodes. Do not use any interface names that is assigned for Kubernetes cluster communication.

The installer displays the cluster configuration details. Enter y to confirm if the information is correct?

16. The installer asks whether you want to configure the Virtual IP. Enter y to create and n to ignore.

17. Press Enter to run the Cluster Server in secure mode.

```
Press [Enter] to continue:

Do you want to grant read access to everyone? [y,n,q,?] (n) y

     1)  Configure the cluster in secure mode
     2)  Configure the cluster in secure mode with FIPS
     b)  Back to previous menu
Select the option you would like to perform [1-2,b,q,?] (1)
```

18. (Optional) Configure SMTP notification.

19. (Optional) Configure the Global Cluster.

20. Configure the InfoScale REST server.

```
Enter the NIC for the REST server to use on <node_name>: [b,q,?] eth0
Is eth0 to be the NIC used by all systems? [y,n,q,b,?] (y)

Enter the virtual IP address for the REST server: [b,q,?]
Enter the NetMask for IP 192.168.2.21: [b,q,?] (255.255.255.0)
Enter a valid port number for the REST server: [b,q,?] (5636)
```

The installer displays the InfoScale REST server Option configuration, Press y to confirm that the information is correct.

21. When prompted, stop the InfoScale Enterprise processes.
22. The installer continues to configure I/O fencing. Enter 1 to configure fencing in preferred majority mode.
23. Check the log file to confirm the installation. The log files, summary file, and response file are saved at: /opt/VRTS/install/logs directory.

Installing Veritas InfoScale with Kubernetes using response files

About response files

The installer script or product installation script generates a response file during any installation, configuration, upgrade, or uninstall procedure. The response file contains the configuration information that you entered during the procedure. When the procedure completes, the installation script displays the location of the response files.

You can use the response file for future installation procedures by invoking an installation script with the **-responsefile** option. The response file passes arguments to the script to automate the installation of that product. You can edit the file to automate installation and configuration of additional systems.

Note: Veritas recommends that you use the response file created by the installer and then edit it as per your requirement.

When you install InfoScale on Kubernetes, you can use the optional **-responsefile** option.

Syntax in the response file

The syntax of the Perl statements that is included in the response file variables varies. It can depend on whether the variables require scalar or list values.

For example, in the case of a string value:

```
$CFG{Scalar_variable}="value";
```

or, in the case of an integer value:

```
$CFG{Scalar_variable}=123;
```

or, in the case of a list:

```
$CFG{List_variable}=["value 1 ", "value 2 ", "value 3 "];
```

Installing Veritas InfoScale using response files

Typically, you can use the response file that the installer generates after you perform Veritas InfoScale installation on a system to install Veritas InfoScale on other systems.

To install Veritas InfoScale on Kubernetes cluster using response files

1. Make sure the systems where you want to install Veritas InfoScale meet the installation requirements.
2. Make sure that the preinstallation tasks are completed.
3. Copy the response file to the system where you want to install Veritas InfoScale.
4. Edit the values of the response file variables as necessary.
5. Mount the product disc and navigate to the directory that contains the installation program.
6. Start the installation from the system to which you copied the response file.

   For example:

   ```
   # ./installer -responsefile /tmp/response_file
   ```

   Where /tmp/response_file is the response file's full path name.
7. Complete the Veritas InfoScale post-installation tasks.

Response file variables to install Veritas InfoScale

Table 7 lists the response file variables that you can define to install Veritas InfoScale.

Table 7: Response file variables for installing Veritas InfoScale

Variable	Description
CFG{opt}{install}	Installs Veritas InfoScale RPMs. Configuration can be performed at a later time using the -configure option. List or scalar: scalar Optional or required: optional
CFG{activecomponent}	Specifies the component for operations like precheck, configure, addnode, install and configure(together). List or scalar: list Optional or required: required
CFG{accepteula}	Specifies whether you agree with the EULA.pdf file on the media. List or scalar: scalar Optional or required: required
CFG{keys}{vxkeyless} CFG{keys}{licensefile}	CFG{keys}{vxkeyless} gives the keyless key to be registered on the system. CFG{keys}{licensefile} gives the absolute file path to the permanent license key to be registered on the system. List of Scalar: List Optional or required: Required.

Variable	Description
CFG{systems}	List of systems on which the product is to be installed or uninstalled. List or scalar: list Optional or required: required
CFG{prod}	Defines the product to be installed or uninstalled. List or scalar: scalar Optional or required: required
CFG{opt}{keyfile}	Defines the location of an ssh keyfile that is used to communicate with all remote systems. List or scalar: scalar Optional or required: optional
CFG{opt}{tmppath}	Defines the location where a working directory is created to store temporary files and the RPMs that are needed during the install. The default location is /opt/VRTStmp. List or scalar: scalar Optional or required: optional
CFG{opt}{rsh}	Defines that rsh must be used instead of ssh as the communication method between systems. List or scalar: scalar Optional or required: optional
CFG{opt}{logpath}	Mentions the location where the log files are to be copied. The default location is /opt/VRTS/install/logs. List or scalar: scalar Optional or required: optional
$CFG{edgeserver_host}	Use this parameter to configure the edge server. Enter telemetry.veritas.com to use the Veritas Cloud Receiver, which is a preconfigured, cloud-based edge server deployed by Veritas. Optional or required: required Note: An edge server is used to collect licensing and platform related information from InfoScale products as part of the Veritas Product Improvement Program. The information collected helps identify how customers deploy and use the product and enables Veritas to manage customer licenses more efficiently.
$CFG{edgeserver_port}	Use this parameter to configure the port number of the edge server. Enter 443, which is the port number used by the Veritas Cloud Receiver. Optional or required: required Note: An edge server is used to collect licensing and platform related information from InfoScale products as part of the Veritas Product Improvement Program. The information

Variable	Description
	collected helps identify how customers deploy and use the product and enables Veritas to manage customer licenses more efficiently.

Response file variables to install Veritas InfoScale on a Kubernetes cluster node

Table 8 lists the response file variables that you can define to install Veritas InfoScale with Kubernetes.

Table 8: Response file variables for installing Veritas InfoScale on a Kubernetes cluster node

Variable	Description
$CFG{REST_server}	Specifies whether to configure REST server or not. List or scalar: scalar Optional or required: optional
$CFG{REST_server_ip}	Specifies IP address to be used by REST server. List or scalar: scalar Optional or required: optional
$CFG{REST_server_netmask}	Specifies Netmask address to be used by REST server. List or scalar: scalar Optional or required: optional
$CFG{REST_server_nic}{all}	Specifies NIC card to be used by all nodes for REST server IP address. List or scalar: scalar Optional or required: optional
$CFG{REST_server_port}	Specifies Port number to be used by REST server IP address. List or scalar: scalar Optional or required: optional
$CFG{container_kube_storage}	Specifies whether to configure Kubernetes with Veritas Container services. List or scalar: scalar Optional or required: optional

Sample response files for Veritas InfoScale installation

```
our %CFG;
$CFG{REST_server}=1;
$CFG{REST_server_ip}="";
```

```
$CFG{REST_server_netmask}="";
$CFG{REST_server_nic}{all}="eth1";
$CFG{REST_server_port}=5636;
$CFG{container_kube_storage}=1;
$CFG{fencing_option}=9;
$CFG{fencingenabled}=1;
$CFG{lltoverudp}=1;
$CFG{pref_majority}=1;
$CFG{prod}="ENTERPRISE743";
1;
```

Completing the post installation tasks

Verifying product installation

To verify the version of the installed product, use the following command:

```
# /opt/VRTS/install/installer -version
```

To find out about the installed RPMs and its versions, use the following command:

```
# /opt/VRTS/install/showversion
```

After every product installation, the installer creates an installation log file and a summary file. The name and location of each file is displayed at the end of a product installation and are always located in the /opt/VRTS/install/logs directory.

Veritas recommends that you keep the files for auditing, debugging, and future use.

The installation log file contains all commands that are executed during the procedure, their output, and the errors generated by the commands.

The summary file contains the results of the installation by the installer or the product installation scripts. The summary includes the list of the RPMs, and the status (success or failure) of each RPM, and information about the processes that were stopped or restarted during the installation. After installation, refer to the summary file to determine whether any processes need to be started.

Setting environment variables

Most of the commands which are used in the installation are present in the /sbin or /usr/sbin directory. Add these directories to your PATH environment variable as necessary.

After installation, Veritas InfoScale commands are in /opt/VRTS/bin. Veritas InfoScale manual pages are stored in /opt/VRTS/man.

Specify /opt/VRTS/bin in your PATH after the path to the standard Linux commands.

Some VCS custom scripts reside in /opt/VRTSvcs/bin. If you want to install a high availability product, add /opt/VRTSvcs/bin to the PATH also.

To invoke the VxFS-specific df, fsdb, ncheck, or umount commands, type the full path name: /opt/VRTS/bin/command.

To set your MANPATH environment variable to include /opt/VRTS/man do the following:

- If you want to use a shell such as sh or bash, enter the following:
  ```
  $ MANPATH=$MANPATH:/opt/VRTS/man; export MANPATH
  ```
- If you want to use a shell such as csh or tcsh, enter the following:

```
% setenv MANPATH $(MANPATH):/opt/VRTS/man
```

- On a Red Hat system, also include the 1m manual page section in the list defined by your MANSECT environment variable.
- If you want to use a shell such as sh or bash, enter the following:
  ```
  $ MANSECT=$MANSECT:1m; export MANSECT
  ```
- If you want to use a shell such as csh or tcsh, enter the following:
  ```
  % setenv MANSECT $(MANSECT):1m
  ```
  If you use the man(1) command to access manual pages, set LC_ALL=C in your shell to ensure that they display correctly.

Managing the Veritas telemetry collector on your server

The Veritas telemetry collector on your server sends telemetry data to the edge server. You can manage the Veritas telemetry collector on each of your servers by using the/opt/VRTSvlic/tele/bin/TelemetryCollector command. See the following table for a list of operations that you can perform to manage the Veritas telemetry collector along with examples of each of the commands.

Table 9: Commands used to manage the collector

Operation	Description

Register an edge server and start the collector	Use the following command to register an edge server with the collector. Note that the collector is automatically started after running this command.
	`/opt/VRTSvlic/tele/bin/TelemetryCollector -start--hostname=<hostname_or_IP> --port=<port_number>`
	• <hostname_or_IP> is the host name or IP address of the edge server you want to register.
	• <port_number> is the port number of the edge server that is used for communication.
	Example:
	`/opt/VRTSvlic/tele/bin/TelemetryCollector -start--hostname=telemetry.veritas.com --port=443`

Start the collector (if the collector is not already running)	Use the following command if you want to start a collector that is not sending telemetry data to the edge server. `/opt/VRTSvlic/tele/bin/TelemetryCollector -start`
Restart the collector (if the collector is already running)	Use the following command to restart the collector that is sending telemetry data to the edge server. `/opt/VRTSvlic/tele/bin/TelemetryCollector -restart`
Check whether the collector is running or not	Use the following command to check the status of the collector on your server. `/opt/VRTSvlic/tele/bin/TelemetryCollector -status`
Check whether the collector is registered with an edge server	Use the following command to check whether the collector is registered with an edge server. `/opt/VRTSvlic/tele/bin/TelemetryCollector-registered`

Operation	Description
Update the host name, IP address, or port number of the edge server	Use the following command to update the host name or IP address and port number of the edge server. `/opt/VRTSvlic/tele/bin/TelemetryCollector -update--hostname=<hostname_or_IP> --port=<port_number>` • <hostname_or_IP> is the host name or IP address of the edge server you want to register. • <port_number> is the port number of the edge server that is used for communication. Examples: • `/opt/VRTSvlic/tele/bin/TelemetryCollector -update--hostname=telemetry.veritas.com --port=443` • `/opt/VRTSvlic/tele/bin/TelemetryCollector -update--port=443`

Configure how often telemetry data is sent to the edge server	Use either of the following commands to define how often you want the collector to send telemetry data to the edge server.
	• To send telemetry data once every month:
	`/opt/VRTSvlic/tele/bin/TelemetryCollector -update--d=<day>`
	• To send telemetry data once every day:
	`/opt/VRTSvlic/tele/bin/TelemetryCollector -update--h=<hour>`
	• To send telemetry data once every week:
	`/opt/VRTSvlic/tele/bin/TelemetryCollector -updatewday=<weekday>`
	Use the following variables in the above commands.
	• <day> is the day of the month that you want the collector to send telemetry data.
	• <hour> is the hour when you want the collector to send the telemetry data.
	• <weekday> is the day of the week that you want the collector to send telemetry data.
	Examples:
	• `/opt/VRTSvlic/tele/bin/TelemetryCollector -update--wday=MON`
	• `/opt/VRTSvlic/tele/bin/TelemetryCollector -update--h=6`
	• `/opt/VRTSvlic/tele/bin/TelemetryCollector -update--d=14`

Next steps after installation

After installation is complete, you can configure a component of your choice.

Table 10 lists the components and the respective Configuration and Upgrade guides that are available.

Table 10: Guides available for configuration

Operation	Description
Storage Foundation and High Availability	See Storage Foundation and High Availability Configuration and Upgrade Guide
Storage Foundation Cluster File System HA	See Storage Foundation Cluster File System High Availability Configuration and Upgrade Guide See Storage Foundation Cluster File System High Availability Administrator's Guide
Cluster Server	See Cluster Server Configuration and Upgrade Guide See Cluster Server Administrator's Guide

Section 4: Configuring InfoScale

Configuring I/O fencing

InfoScale uses I/O fencing to guarantee data protection and to provide persistent storage to Kubernetes. Fencing ensures that data protection is the highest priority and stops running the systems when necessary to ensure that the systems cannot starts services when a split-brain condition is encountered. InfoScale checks for the connectivity with each peer nodes periodically while Kubernetes checks it for the master to worker nodes.

The Kubernetes cluster performs failover or restart of applications for the nodes that have reached a NotReady state in the Kubernetes cluster. If an application is configured as a statefulset pod, Kubernetes by design stops the failover of such application pods till the node becomes active again. In such scenarios, InfoScale uses the fencing module to ensures that the application pods running on such unreachable nodes cannot access the persistent storage so that Kubernetes can restart these pods on the active cluster again without the risk of any data corruption.

When InfoScale cluster is deployed with Kubernetes, Kubernetes does not perform failover of statefulset pods. Therefore, InfoScale uses Kube-fencing, a custom fencing controller to provide the fencing infrastructure. The custom Kube-fencing controller interacts with the InfoScale fencing driver and enables the failover in Kubernetes in case of network split. The controller runs an agent that ensures that InfoScale fences out the persistent storage and performs the pod failover for the fenced-out node. It also ensures that the fencing decisions between the InfoScale I/O fencing compliments the fencing decisions by the fencing controller.

For deployment in containerized environments, when you install InfoScale using the product installer, the fencing module is atomically installed and configured in preferred majority mode. In case of network split, the I/O fencing module take fencing decisions based on the number and assigned weightage of the Kubernetes master nodes in a sub-cluster.

To configure I/O fencing in a Kubernetes environment, the following prerequisites must be met:

- Kubernetes cluster is already installed
- Kubernetes Time to Live controller is enabled
- The hostnames of InfoScale nodes must exactly match the FQDN of the Kubernetes nodes

Configuring I/O fencing manually

Perform the following steps to configure I/O fencing manually in a Kubernetes environment.

1. Install the Vxfen package. You can manually install it from the Veritas InfoScale ISO image.
   ```
   # rpm -ivh VRTSvxfen
   ```
2. Prepare the VxFEN configuration file.

   Note: You can use the sample configuration file available in the VRTSvxfen rpm package to configure fencing in Kubernetes environment.
   ```
   # cp /etc/vxfen.d/vxfen_pref_majority /etc/vxfenmode
   ```
3. Start the vxfen service
   ```
   # systemctl start vxfen.service
   ```
4. Verify that the service is started properly
   ```
   # systemctl status vxfen.service
   ```
5. Run the following command on all the nodes in Kubernetes cluster to set fencing specific annotations for Kubernetes node objects.
   ```
   # kubectl annotate --overwrite node <node_name> fencing/enabled=true
   ```
6. Apply the kube-fencing specific yamls on one of the Kubernetes master nodes in the cluster.
   ```
   # kubectl apply -f /etc/vxfen.d/K8/kube-fencing.yaml
   ```

```
# kubectl apply -f /etc/vxfen.d/K8/vxfen_agent.yaml
```

7. Load the vxfen container image used by fencing pods on all the nodes in kubernetes cluster
   ```
   # /etc/vxfen.d/K8/critools.sh -l /etc/vxfen.d/K8/vxfenimage.tar
   ```
8. Assign weights to each cluster node based on their role in the Kubernetes cluster.

   To assign weight to a Kubernetes master node:

   a. Enable fencing in VCS configuration by editing the main.cf file and adding the UseFence entry to cluster resource as:
      ```
      cluster esf (
              SecureClus = 1
              HacliUserLevel = COMMANDROOT
              UseFence = SCSI3
              )
      ```
   b. Start the VCS service in the cluster.
      ```
      # systemctl start vcs.service
      ```
   c. Make the VCS configuration writable.
      ```
      # haconf -makerw
      ```
   d. Set the value of the cluster-level attribute PreferredFencingPolicy as System in HAD.
      ```
      # haclus -modify PreferredFencingPolicy System
      ```
   e. Run the following command to assign weight to the Kubernetes master node using its hostname.
      ```
      # hasys -modify sys1 FencingWeight 100
      ```
      For example, in a two-node cluster, where you want to assign sys1 five times more weight compared to sys2, run the following commands:
      ```
      # hasys -modify sys1 FencingWeight 50
      # hasys -modify sys2 FencingWeight 10
      ```
   f. Repeat step e for every master node in the cluster.
   g. Save the VCS configuration.
      ```
      # haconf -dump -makero
      ```
9. Run the following command to verify that the weights are assigned in vxfen driver.
   ```
   vxfenconfig -a
   ```
   ```
   Fencing Mode: PREF_MAJORITY
   Race Policy : count

            Node  Weight
               0      100
               1      0
               2      0

   VXFEN vxfenconfig NOTICE Cluster not configured to use node weights for fencing.
   ```

CSI plugin deployment on Kubernetes

CSI is a standardized mechanism for Container Orchestration Systems (COs) to expose arbitrary storage systems to their containerized workloads. InfoScale CSI plugin is used to provide persistent storage to Kubernetes. InfoScale CSI also support creation of storage classes for high availability, performance, capacity, and data security. It also supports online expansion of capacity as well as snapshot and clone functionality.

InfoScale CSI is automatically deployed while installing InfoScale Enterprise version 7.4.3 and later with Kubernetes. During the CSI deployment using the common product installer (CPI), the installer equips InfoScale with a variety of Yet Another Markup Language (YAML) files for creating storage classes, persistent volumes, and persistent volume claims. You can also create YAML files specific to your requirements.

You can use these YAML files for:

- Dynamic provisioning of volumes
- Static provisioning of volumes
- Snapshot provisioning (Creating volume snapshots)
- Creating volume clones

For more information on the sample YAML files provided with InfoScale package, see Appendix: YAML files.

InfoScale CSI supports static and dynamic provisioning of volumes on shared storage (CVM) as well as shared nothing storage (FSS).

Note: Only one disk group is supported for all CSI operations, and the same disk group is used throughout the CSI plugin lifecycle. You must create the disk group in advance and ensure that it has the same name that was provided during InfoScale installation using the installer.

An application container requests for the required storage through a Persistent Volume claim (PVC). The PVC uses the storage class to identify and provision the Persistent Volume that belongs to the storage class. Once the volume is created, a Persistent Volume object is created and is bound to the PVC, and persistent storage is made available to the application.

While provisioning volumes, the InfoScale CSI plugin supports the following access modes that determines how the volumes can be mounted:

- ReadWriteOnce (RWO) -- the volume can be mounted as read-write by a single node.
- ReadOnlyMany (ROX) -- the volume can be mounted read-only by many nodes.
- ReadWriteMany (RWX) -- the volume can be mounted as read-write by many nodes.

Note that the permission in a Persistent Volume Claim is per node and not per pod. For example, a PVC with RWO mode does not prevent mounting same volume in more than one pod on same node.

Static provisioning

You can use static provisioning if you want to make the existing persistent storage objects available to the cluster. You can statically provision a volume over shared storage (CVM) and shared nothing (FSS) storage.

Static provisioning allows cluster administrators to make existing storage objects available to a cluster. To use static provisioning, you must know the details of the storage object, its supported configurations, and mount options. To make existing storage available to a cluster user, you must manually create a Persistent Volume, and a Persistent Volume Claim before referencing the storage in a pod.

Note: You must ensure that the VxFS file system is created before provisioning the volumes statically. If the VxFS file system does not exist, you must create it manually using the mkfs command.

1. Create a Storage Class using a yaml file.
   ```
   # kubectl create -f csi-infoscale-sc.yaml
   ```
2. Define the Persistent Volume object and specify the existing VxVM volume name in the volumeHandle attribute.
   `csi-static-pv.yaml`

```
---
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: csi-infoscale-pv
  annotations:
    pv.kubernetes.io/provisioned-by: org.veritas.infoscale
spec:
  storageClassName: csi-infoscale-sc
  persistentVolumeReclaimPolicy: Delete
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  csi:
    driver: org.veritas.infoscale
    # Please provide pre-provisioned Infoscale volume name.
    volumeHandle: <existing_VxVM_volume_name>
    fsType: vxfs
```

3. Create a Persistent Volume using the yaml.
   `# kubectl create -f csi-infoscale-static-pv.yaml`
4. Define the Persistent Volume Claim (PVC) with appropriate access mode and storage capacity.
   `csi-static-pvc.yaml`

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

5. Create a Persistent Volume Claim using the yaml. This PVC automatically gets bound with the newly created PV.
   `# kubectl create -f csi-infoscale-static-pvc.yaml`
6. Update the application yaml file and specify the persistent Volume Claim name.
   `mysql-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
```

```
  matchLabels:
    app: mysql
template:
  metadata:
    labels:
      app: mysql
  spec:
    containers:
      - name: mysql
        image: mysql:latest
        ports:
          - containerPort: 3306
        volumeMounts:
          - mountPath: "/var/lib/mysql"
            name: mysql-data
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: root123
    volumes:
      - name: mysql-data
        persistentVolumeClaim:
          claimName: csi-infoscale-pvc
```

7. Create the application pod.
   ```
   # kubectl create -f csi-mysql-app.yaml
   ```
8. Check that old data exists on the persistent volume.
   ```
   # kubectl get pods | grep mysql
   # kubectl exec -it mysql-deployment<id> -- mysql -uroot -pRoot12345!
   ```

Dynamic provisioning

You can dynamically provision a volume over shared storage (CVM) and shared nothing (FSS) storage.

In dynamic provisioning, you must create a Storage Class that define the storage provisioner and the required parameters in the storage class yaml file and create the Persistent Volume Claim. The Pod references the Storage Class through an existing Persistent Volume Claim and allocate storage for the requesting Pod dynamically.

You can allocate storage to pods dynamically and also reclaim the storage when the previously provisioned storage is available for other applications to use. You can also create encrypted volumes and resize an existing volume using the Persistent Volume Claim (PVC) object.

Allocating storage dynamically to container workloads

1. Create a Storage Class using a yaml file.
   ```
   # kubectl create -f csi-infoscale-sc.yaml
   ```
2. Define the Persistent Volume Claim and specify the appropriate Storage Class, access mode, and the required storage size.
   ```
   csi-dynamic-pvc.yaml
   ```

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-infoscale-pvc
spec:
  storageClassName: csi-infoscale-sc
  accessModes:
    - ReadWriteMany

  resources:
    requests:
      storage: 5Gi
```

3. Create a Persistent Volume Claim using the yaml.
   `# kubectl create -f csi-dynamic-pvc.yaml`
4. Update the application yaml file and specify the persistent Volume Claim name.
   `csi-mysql-app.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:latest
          ports:
            - containerPort: 3306
          volumeMounts:
            - mountPath: "/var/lib/mysql"
              name: mysql-data
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: root123
      volumes:
        - name: mysql-data
          persistentVolumeClaim:
```

```
            claimName: csi-infoscale-pvc
```

5. Create the application pod.

   ```
   # kubectl create -f csi-mysql-app.yaml
   ```

   Once the pod is created, start using the InfoScale PVC as a Persistent Storage.

Reclaiming provisioned storage

When a previously provisioned storage is no longer required by an application, you can delete the corresponding PVC objects from the APIs and reclaim the storage for other applications to use. The reclaim policy for a Persistent Volume states what action the cluster should take on the volume after it is released from the PVC.

You can use the following command to delete a PVC:

```
#kubectl delete pvc <pvc_name>
```

InfoScale supports the following reclaim policies. You must specify the reclaim policy while creating a storage class for dynamic provisioning.

- **Retain**: Indicates that the Persistent Volume should be reclaimed manually.
- **Delete**: (Default) Indicates that the Persistent Volume and the associated storage gets automatically deleted when the PVC is deleted.

For more information on the reclaim policies, see Kubernetes documentation.

Creating encrypted volumes

VxVM provides advanced security for data at rest through encryption of VxVM data volumes. Encryption is a technology that converts data or information into a code that can be decrypted only by authorized users.

Veritas InfoScale supports encrypting data at rest and data encryption over wire. It helps you to take regular backups of the encrypted volumes. See the Storage Foundation Cluster File System High Availability 7.4.3 Administrator's Guide - Linux.

You can encrypt VxVM data volumes to:

- Protect sensitive data from unauthorized access
- Retire disks from use or ship them for replacement without the overhead of secure wiping of content

In a Kubernetes environment, you can create an encrypted volume by setting the encryption attribute to true in the Storage Class yaml file. The Storage Class YAML file templates are stored in the `/etc/vx/csi/deployment/kubernetes/StorageClass_templates` directory.

To create an encrypted volume using the YAML file:

1. Ensure that you configure the KMS server on all InfoScale nodes to use the encryption feature.

   Note: InfoScale deployment with Kubernetes only support volume encryption using the Key Management Server (KMS).

2. Update the Storage Class file definition and set the encryption attribute to true.

   `csi-infoscale-secure-sc.yaml`

   ```
   ---
   apiVersion: storage.k8s.io/v1
   kind: StorageClass
   metadata:
   name: csi-infoscale-secure-sc
   annotations:
   storageclass.kubernetes.io/is-default-class: "false"
   ```

```
provisioner: org.veritas.infoscale
reclaimPolicy: Delete
allowVolumeExpansion: true
parameters:
fstype: vxfs

# (optional) Specifies a volume layout type.
# Supported layouts: stripe, mirror, stripe-mirror, mirror-stripe, concat, concat-mirror, mirror-concat
layout: "mirror"

# (optional) Specifies the number of disk or host failures a storage object can tolerate.
faultTolerance: "1"

# (optional) Specifies whether to store encrypted data on disks or not.
# Valid values are true or false
encryption: "true"
```

3. Create a Storage Class using a yaml file.

   ```
   # kubectl create -f csi-infoscale-secure-sc.yaml
   ```

4. Define the Persistent Volume Claim and specify the Storage Class, access mode, and the required storage size.

   `csi-dynamic-pvc.yaml`

   ```
   ---
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: csi-infoscale-pvc
   spec:
     storageClassName: csi-infoscale-secure-sc
     accessModes:
       - ReadWriteMany

     resources:
       requests:
         storage: 5Gi
   ```

5. Create a Persistent Volume Claim using the static or dynamic provisioning yaml.

   ```
   # kubectl create -f csi-dynamic-pvc.yaml
   ```

6. Once the volume is provisioned, check if the created volume is tagged as encrypted.

   ```
   # vxencrypt list
   ```

   An output similar to this is displayed.

   ```
   Disk group: dg_name
   VOLUME STATUS
   <volume_name> encrypted
   ```

7. Once the PVC is created, create the container application pod, run the application, and access the encrypted volume.

Resizing Persistent Volumes (CSI volume expansion)

Using the persistent volume expansion feature, you can easily expand the storage capacity of a persistent volume by just updating the Persistent Volume Claim storage specification. However, to use this feature, you must set the allowVolumeExpansion attribute to true in their StorageClass object. Only those PVCs created using such Storage Class allows volume expansion. When the storage attribute of such a PVC object is updated, Kubernetes interpret it as a change request and triggers automatic volume resizing.

The following sample Storage Class yaml shows the allowVolumeExpansion attribute definition.

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

However, while performing the volume expansion operation, you must note the following:

- Ensure that the PVC is in use by some application pod while performing resize operation. InfoScale supports dynamic volume expansion in 'Online' mode.
- Do not perform the volume expansion operation from the host backend. In such case, Kubernetes is not aware of these changes and updated volume size is not reflected in the PV and PVC objects.
- InfoScale does not support the shrinking of persistent volume as Kubernetes does not support it.

Once the volume is provisioned, create the container application pod, run the application, and access the volume. If the volume is full and must be resized, use one of the following ways:

- Edit the Persistent Volume Claim
- Use the kubectl patch pvc command

Note: Resize operation on volume is not supported when it is provisioned in ReadOnlyMany mode.

Resizing a Persistent Volume by editing the Persistent Volume Claim

1. Find the Persistent Volume Claim to resize.

   ```
   # kubectl get pvc
   ```

   Output similar to this is displayed:

   ```
   NAME            STATUS     VOLUME    CAPACITY  ACCESS MODES  STORAGECLASS      AGE
   csi-infoscale-pvc Bound  pvc-<id>    1Gi       RWX          csi-infoscale-sc    32m
   ```

2. To resize the storage capacity, edit the PVC.

   ```
   # kubectl edit pvc csi-infoscale-pvc
   ```

   From your text editor, change the storage capacity to the required value. For example, from 1Gi to 10Gi

3. Check the status of the Persistent Volume Claim and Persistent Volume to verify if the size is updated.

   ```
   # kubectl get pvc
   ```

Output similar to this is displayed:

```
NAME              STATUS   VOLUME    CAPACITY     ACCESS MODES  STORAGECLASS     AGE
csi-infoscale-pvc Bound    pvc-<id> 10Gi       RWX        csi-infoscale-sc   32m
```

Resizing a Persistent Volume using the 'patch pvc' command

1. Find the Persistent Volume Claim to resize.

   ```
   # kubectl get pvc
   ```

   Output similar to this is displayed:

   ```
   NAME             STATUS   VOLUME    CAPACITY    ACCESS MODES  STORAGECLASS      AGE
   csi-infoscale-pvc   Bound pvc-<id>  1Gi         RWX         csi-infoscale-sc    32m
   ```

2. To resize the storage capacity to the required value, for example, from 1Gi to 10Gi, run the following command.

   ```
   # kubectl patch pvc csi-infoscale-pvc --patch '{"spec": {"resources": {"requests":
   {"storage": "10Gi"}}}}'
   ```

3. Check the status of the Persistent Volume Claim and Persistent Volume to verify if the size is updated.

   ```
   # kubectl get pvc
   ```

   Output similar to this is displayed:

   ```
   NAME              STATUS   VOLUME      CAPACITY  ACCESS MODES STORAGECLASS      AGE
   csi-infoscale-pvc   Bound  pvc-<id> 10Gi       RWX         csi-infoscale-sc    32m
   ```

Snapshot provisioning (Creating volume snapshots)

Volume snapshot represents a point-in-time and space-optimized copy of volume on storage system. The InfoScale CSI Plugin supports snapshot provisioning. You can create one or more snapshots of Persistent Volume that is provisioned dynamically or statically. You can also restore a Snapshot to reinstate the volume contents on a completely new Persistent Volume that you want to provision. The snapshots can also be consumed directly as PVC through static provisioning.

Note: For using the point-in-time copies, Veritas recommends that you:

- Use the space-optimized snapshots for read-intensive applications that runs on top of either a source Persistent Volume or a snapshot copy. You can use full-instant snapshots for the write-intensive applications.
- Use the Volume Clones feature for write- intensive applications. The volume Clones makes the exact copy of a Persistent volume immediately available for the read, write, and update operations.

To create a snapshot, you must create the following objects by using the yaml files:

- A **VolumeSnapshotContent** is a cluster resource to create a snapshot of a volume in the cluster that is provisioned by an administrator. This resource is similar to a PersistentVolume.
- A **VolumeSnapshot** is a cluster resource to create a snapshot of a volume in the cluster that is provisioned by a user. This resource is similar to a PersistentVolumeClaim.
- A **VolumeSnapshotClass** describe the storage classes when provisioning a volume snapshot. The **VolumeSnapshotClass** acts as a template for creating a snapshot and includes attributes like the type of snapshot, synchronization parameters, and other configuration parameters. The VolumeSnapshotClass yaml files are copied to the /etc/vx/csi/deployment/kubernetes/SnapshotClass_templates/ directory during the InfoScale installation.

IMPORTANT:

- The VolumeSnapshot, VolumeSnapshotContent, and VolumeSnapshotClass API objects are Custom Resource

Definitions (CRDs) and not a part of the core API.

- In the beta version of VolumeSnapshot, you must deploy a snapshot controller into the control plane and deploy csi-snapshotter, a sidecar helper container, along with the CSI driver.
- Although the Kubernetes distribution deploys the CRDs and snapshot controller installations, you must deploy the csi-snapshotter sidecar container as a part of the CSI driver controller pod.

Dynamic provisioning of a snapshot

1. Define the VolumeSnapshotClass object using the yaml and specify the deletionPolicy and snapType.

   `csi-infoscale-snapclass.yaml`

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: csi-infoscale-snapclass
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true"
driver: org.veritas.infoscale
deletionPolicy: Delete
#parameters:
  # (optional) Specifies the type of the snapshot to be created.
  # If omitted, InfoScale by default creates space-optimized snapshot.
  # Supported values: space-optimized, full-instant
  # snapType: space-optimized

  # (optional) Specifies the size of the cache volume to be created for space-
optimized snapshots.
  # If omitted, InfoScale internally chooses the cacheSize as 30% of orignial
volume size.
  # cacheSize: 500m
```

2. Create Volume Snapshot Class

```
# kubectl create -f csi-infoscale-snapclass.yaml
```

3. Define the Volume Snapshot.

   `csi-dynamic-snapshot.yaml`

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: csi-dynamic-snapshot
spec:
  volumeSnapshotClassName: csi-infoscale-snapclass
  source:
    persistentVolumeClaimName: csi-infoscale-pvc
```

4. Create Volume Snapshot

```
# kubectl create -f csi-dynamic-snapshot.yaml
```

   On successful creation of a snapshot, the corresponding volume snapshot content is created and bound to the volume

Snapshot object.

Static provisioning of an existing snapshot

1. Define the volume snapshot content object using the yaml file and specify the snapshotHandle.
   `csi-static-snapshot-content.yaml`

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotContent
metadata:
name: csi-static-snapshot-content
spec:
deletionPolicy: Retain
driver: org.veritas.infoscale
source:
# Provide pre-provisioned Infoscale snapshot volume name
snapshotHandle: testSnapVol
volumeSnapshotRef:
name: csi-static-snapshot
namespace: default
```

2. Define the volume snapshot object using the yaml and specify the volumeSnapshotContentName.
   `csi-static-snapshot.yaml`

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
name: csi-static-snapshot
spec:
volumeSnapshotClassName: csi-infoscale-snapclass
source:
volumeSnapshotContentName: csi-static-snapshot-content
```

3. Create Volume Snapshot

   ```
   # kubectl create -f csi-static-snapshot.yaml
   ```

   On successful creation of a snapshot, the corresponding volume snapshot content is created and bound to the volume Snapshot object.

Using a snapshot

You can use the snapshots created using the VolumeSnapshot request by restoring them to a new PVC and provisioning that PVC with the pre-populated data from snapshot to an application pod.

You can also use the snapshot volumes as static Persistent Volumes by specifying the snapshot volume name as a value for the volumeHandle parameter while provisioning a static PV.

Restoring a snapshot to new PVC

If you want to use and update a point-in-time copy of the application data, you can restore the snapshot of that application's persistent volume to a new persistent volume that represents the previous state described by the snapshot. To restore a volume from a snapshot, you must specify the name of the VolumeSnapshot object that you want to restore as the value of the dataSource attribute.

Note: While restoring a snapshot or a clone to a new PVC, you must specify the exact same storage details as specified in the source PVC.

1. Define a Persistent Volume Claim object using the yaml and specify the name of the VolumeSnapshot object that you want to restore in the dataSource attribute:
   `csi-dynamic-snapshot-restore.yaml`

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: csi-infoscale-snapshot-restore
spec:
storageClassName: csi-infoscale-sc
accessModes:
- ReadWriteMany
resources:
requests:
storage: 5Gi
dataSource:
name: csi-dynamic-snapshot
kind: VolumeSnapshot
apiGroup: snapshot.storage.k8s.io
```

2. Create the Persistent Volume Claim.

```
# kubectl create -f csi-dynamic-snapshot-restore.yaml
```

Deleting a volume snapshot

You can delete one or more snapshots by deleting the volume snapshot object associated with snapshot. If you set the DeletionPolicy to Delete while defining the snapshot object, then the underlying storage snapshot is automatically deleted when the VolumeSnapshotContent object is deleted.

Use the following command to delete the VolumeSnapshot object:

```
# kubectl delete volumesnapshot csi-dynamic-snapshot
```

Note:

- For space-optimized snapshots, InfoScale maintains the association between the source PVC and the snapshot volume. Therefore, you must delete the snapshot objects before deleting the source PVC.
- For full-instant snapshots, you can delete the source PVC before deleting the snapshot object once the synchronization between these two is completed.

Managing InfoScale volume snapshots with Velero

Velero is a backup and recovery solution that assists in backing up and restoring the applications and their corresponding persistent volumes in a Kubernetes environment.

You can integrate InfoScale CSI plugin with Velero to backup and restore CSI-backed volumes across clusters.

The following example shows how to configure and use Velero with InfoScale CSI plugin snapshots feature. This example uses MinIO object storage server for storing objects metadata.

Setting up Velero with InfoScale CSI.

Prerequisite:

- Download and install Velero CLI. For more information, see Velero documentation.

To configure Velero:

1. Set up the InfoScale CSI environment.

   See CSI plugin deployment on Kubernetes.
2. Set up the MinIO server.

   The 00-minio-deployment.yaml file to set up the MinIO server is included in the Velero package. You must edit the IP addresses and ports in the yaml as required.

   ```
   #kubectl apply -f 00-minio-deployment.yaml
   ```
3. Create the Velero secret file with the credentials to access the MinIO server.

   ```
   [default]
   aws_access_key_id=<user_id>
   aws_secret_access_key=<passowrd>
   ```
4. Install Velero by running the below command

   ```
   #velero install \
   --provider aws \
   --features=EnableCSI \
   --plugins=velero/velero-plugin-for-csi:v0.1.0,velero/velero-plugin-for-aws:v1.0.0 \
   --bucket velero \
   --secret-file ./credentials-velero \
   --use-volume-snapshots=True \
   --backup-location-config
   region=minio,s3ForcePathStyle="true",s3Url=http://minio.velero.svc:9000,publicUrl=h
   ttp://<ip>:<port> \
   --snapshot-location-config region=default,profile=default
   ```
5. Deploy the application that uses the CSI backed InfoScale volumes.
6. Create a VolumeSnapshotClass for the CSI backed volumes using the csi-infoscale-snapclass yaml.

   ```
   # csi-infoscale-snapclass.yaml
   ```

   ```
   ---
   apiVersion: snapshot.storage.k8s.io/v1beta1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-infoscale-snapclass
     annotations:
       snapshot.storage.kubernetes.io/is-default-class: "false"
   driver: org.veritas.infoscale
   ```

```
deletionPolicy: Retain
parameters:
  snapType: full-instant

 Create a VolumeSnapshotClass
```

```
# kubectl create -f csi-infoscale-snapclass.yaml
```

Taking the Velero backup

After you configure the Velero setup, you can back up all objects in your cluster, or you can filter objects by type, namespace, and label. For more information, see Velero documentation.

Use the **velero backup create** command to back up applications that are using the CSI volumes.

For example, to take a backup of a namespae run the following command:

```
# velero backup create <backup_name> --include-namespaces=<namespace_name> -wait
```

Note: When you take a backup using Velero, the PVCs of the CSI Volumes are backed up as snapshots on the on-premises InfoScale host.

Creating a schedule for backup

The schedule operation allows you to back up your data at specified periodic intervals. The first backup is performed when the schedule is created, and subsequent backups happen at the scheduled interval.

Scheduled backups are saved with the name <SCHEDULE NAME>-<TIMESTAMP>, where <TIMESTAMP> is formatted as YYYYMMDDhhmmss.

In a Kubernetes environment, the scheduled backup operation create snapshots of the CSI-backed volumes on a pre-defined time interval.

For example, use the following sample yaml file to create backup schedules of the nginx-app namespace after every 30 minutes that has a validity of 2 hours.

```
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: daily
  namespace: velero
spec:
  schedule: "*/30 * * * *"
  template:
    hooks: {}
    includedNamespaces:
    -   nginx-app
    ttl: 02h00m0s
```

Restoring from the Velero backup

The restore operation allows you to restore all objects and persistent volumes from a previously created backup. You can also restore only a subset of objects and persistent volumes. For more information, see Velero documentation.

The default name of a restore is <BACKUP NAME>-<TIMESTAMP>, where <TIMESTAMP> is formatted as YYYYMMDDhhmmss. You can also specify a custom name.

Use the **velero restore create** command to restore the Kubernetes objects and InfoScale CSI volumes from the previously created backup.

For example:

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

Volume cloning

The CSI volume clone feature duplicates an existing Persistent Volume at given point in time. Cloning creates an exact duplicate of the specified volume on the backend rather than creating a new empty volume. When a clone is created, it is an independent object that can be used as any other PVC. The data of the cloned volume is also in sync with the data of the original dataSource PVC. The cloned volume can be consumed, cloned, snapshotted, or deleted without affecting the original dataSource PVC.

IMPORTANT

You can clone a PVC only when the following conditions are met:

- The source and destination PVCs are in the same namespace
- The source and destination storage class are the same

Creating volume clones

The cloning feature enables you to specify an existing PVC as a dataSource while creating a new PVC. However, to create a clone, the following prerequisites must be met:

- The source PVC is bound and available for use
- A valid Storage class is available
- The source PVC is created using the InfoScale CSI driver that supports volume cloning

To clone a PVC from an existing PVC:

1. Identify the PVC that you want to clone.
   ```
   # kubectl get pvc
   ```
2. Define the PersistentVolumeClaim object using the yaml and specify the name of the PVC object to use as source
   `csi-dynamic-volume-clone.yaml`

```
---
  kind: PersistentVolumeClaim
  apiVersion: v1
  metadata:
    name: csi-infoscale-volume-clone
  spec:
    storageClassName: csi-infoscale-sc
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 1Gi
    dataSource:
      kind: PersistentVolumeClaim
      name: csi-infoscale-pvc
```

3. Create a volume clone

   ```
   # kubectl create -f csi-dynamic-volume-clone.yaml
   ```

   On successful creation of a clone, it is pre-populated with the data from the specified PVC dataSource volume.

Deleting a volume clone

To delete a volume clone, use the following command:

```
# kubectl delete pvc csi-infoscale-volume-clone
```

Verify that the volume clone is deleted and not displayed in the output of the `# kubectl get pvc` command.

ConfigMaps and Kubernetes Secrets

Kubernetes provides the ConfigMaps and Secrets utilities to facilitate the configuration of an application with ease. They add flexibility by separating the configuration information of an individual container instance from the container image. Simply put, Secrets and ConfigMaps contain configuration data that can be used by the pods as config files, command-line arguments, and environment variables. ConfigMaps and Secrets are files with one or more key-value sets that are executed using the Kubernetes API. For example, you can set the MYSQL_ROOT_PASSWORD in a Secret and add it to the container as an environment variable. You can store the configuration files in a ConfigMap and mount it into the container as a file on startup.

ConfigMaps

You can use a ConfigMap to store non-confidential data in key-value pairs, such as file paths and file names consumed by the pods through the sidecar container. Veritas InfoScale provides an optional VCS_CONFIGMAP environment variable that a user can use to specify a file name either as an absolute file path or just a file name. Based on the value of this environment variable, the VCS sidecar container discovers a configuration file inside a container. The configuration file that is mounted inside the sidecar container contains the configuration information you want to make available to the deployment.

You can customize the InfoScale VCS configuration by using the VCS_CONFIGMAP environment variable in a ConfigMap.

For more information on the environment variables, see Configuring parameters between VCS and Kubernetes.

Creating a ConfigMap from a configuration file

Note: Although Kubernetes provides multiple ways to create a ConfigMap, InfoScale supports the ConfigMaps that are created using a configuration file.

To create a ConfigMap from a configuration file:

1. Verify that the content of the configuration file that you want to use for creating the ConfigMap follows the InfoScale VCS Config file format.

   ```
   # cat <config_file>.conf
   ```

   Note: While writing a config file, user must follow the below instructions:

   - Specify the application resources that you want to monitor at the top in the config file.
   - Ensure that while specifying InfoScale VCS attributes in the config file, you specify the attributes only in the VCS_AppType_VcsAttrName format. For example, set VCS_Mount_BlockDevice_1=tmpfs to configure BlockDevice attribute for a Mount resource
   - Separate each resource definitions and its attributes by a "----------"

   For example:

   ```
   # cat config-file.conf
   VCS_MySQL_MySQLAdmin=mysqladmin
   VCS_MySQL_BaseDir=/usr
   ```

```
VCS_MySQL_DataDir=/var/lib/mysql
-----------------------------------------------------
VCS_Mount_BlockDevice_1=tmpfs
VCS_Mount_MountPoint_1=/sys/firmware
VCS_Mount_FSType_1=tmpfs
VCS_Mount_FsckOpt_1=%-n
-----------------------------------------------------
VCS_NIC_Device=eth0
-----------------------------------------------------
```

2. To Create the ConfigMap and import the configuration file into a ConfigMap, run the following command:

   `# kubectl create configmap <configmap_name> --from-file=<config_file>.conf`

   For example:

   `# kubectl create configmap mysql-configmap --from-file=config-file.conf`

3. Validate the content of the ConfigMap.

   `# kubectl describe configmap <configmap_name>`

   For example:

   `# kubectl describe configmap mysql-configmap`

   The content of the ConfigMap is displayed.

```
Name:         mysql-configmap
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
config-file.conf:
----
VCS_MySQL_MySQLAdmin=mysqladmin
VCS_MySQL_BaseDir=/usr
VCS_MySQL_DataDir=/var/lib/mysql
-----------------------------------------------------------
VCS_Mount_BlockDevice_1=tmpfs
VCS_Mount_MountPoint_1=/sys/firmwareVCS_Mount_FSType_1=tmpfs
VCS_Mount_FsckOpt_1=%-n
-----------------------------------------------------------
VCS_NIC_Device=eth0
-----------------------------------------------------------
Events:  <none>
```

Kubernetes secrets

Kubernetes secrets let you store and manage sensitive information such as password, authorization tokens, and SSH keys. Though Kubernetes creates some secret files that contains credentials required for accessing the API. You can also create your own secret files. These secret files are encrypted to ensure security.

Multiple Pods can reference the same secret. Secrets can be mounted as data volumes or exposed as environment variables to be used by a container in a Pod. Secrets can also be used by other parts of the system, without being directly exposed to the Pod. For example, Secrets can hold credentials that other parts of the system should use to interact with external systems.

Creating secrets

Note: Although Kubernetes provides multiple ways to create a secret, InfoScale support the secrets that are created using the literal values. In this method, you can create a single secret that package one or more key-value pairs.

InfoScale VCS recommends that you should include the sensitive information required by all containers in the POD in one secret. The key from the secret becomes the environment variable name in the Pod. Therefore, you can use each key as a key or as an environment variable.

For example, MySQL application container requires the MYSQL_PASSWORD environment variable. Therefore, in YAML file for application container, the MYSQL_PASSWORD key is used as an environment variable. In sidecar containers, VCS uses secret as a volume mounted file. In such case, you can reuse MYSQL_PASSWORD as a key for the volume that you plan to mount inside the sidecar container as a secret file.

Creating a secret from a lateral value

To create a secret file from one or more key-value data pairs, run the following command:

```
# kubectl create secret generic <secret_name> --from-literal=key1=value1 --from-
literal=key2=value2
```

For example:

```
# kubectl create secret generic my-secret --from-literal=MYSQL_PASSWORD='Vcs12345' --
from-literal=MYSQL_ROOT_PASSWORD='Vcs12345' --from-literal=MYSQL_USER='mysqladmin' --
from-literal=MYSQL_DATABASE='mysql'
```

Verify that the secret is created.

```
# kubectl describe secret <secret_name>
```

Configure a Pod to use ConfigMaps and secrets

You can use the configuration data from a ConfigMaps and the key-value pairs of sensitive information from a Secrets as container environment variables .

You can do so by following the following steps:

1. Create the required ConfigMaps and Secrets with the required configuration data
2. Update the container application pod definition in the yaml file
3. Create the container application pod.

Configurations parameters between VCS and Kubernetes

InfoScale allows you to provide environment variables while defining Pods. You can use the set the following environment variables while configuring application pods.

Table 11: Mandatory environment variables used by InfoScale in a Kubernetes environment

Environment variable name	Description
VCS_RUN_IN_SIDECAR_CONTAINER	Specifies whether to run InfoScale VCS inside the sidecar container.

Environment variable name	Description
	Set the value of this variable to 1 to run InfoScale VCS inside the sidecar container.
VCS_RUN_IN_APP_CONTAINER	Specifies whether to run InfoScale VCS inside the application container. Set the value of this variable to 1 to run InfoScale VCS inside the application container.
VCS_SECRET_MySQLAdminPasswd	Contains the absolute path for a secret file that is mounted inside a sidecar container and contains the MySQLAdminPasswd attribute value. Examples: If you set VCS_SECRET_MySQLAdminPasswd = "/var/VRTSvcs/container/secret/MySQLAdminPasswd", then, the file name and the file mount path values specified for the environment variable must match the values specified for the attributes in a secret file. That is, .spec.volumes.secret.items.path = "MySQLAdminPasswd" .spec.container.voulumeMounts.mountPath = "/var/VRTSvcs/container/secret " If a secret is created with key-value pair, that is using a literal as: --from-literal=MYSQL_PASSWORD='Vcs12345' Then, you can reuse the MYSQL_PASSWORD key for the secret volume as: .spec.volumes.secret.items.key: MYSQL_PASSWORD

Table 12: Optional environment variables used by InfoScale in a Kubernetes environment

Environment variable name	Description
VCS_CONFIGMAP	Enables the sidecar container to locate the ConfigMap. You can specify the ConfigMap file location as a value of this parameter in one of the following ways: **As an absolute path:** For example: If you set the variable as: VCS_CONFIGMAP = "/tmp/configmaps/data/my-config.conf", you must ensure that: • The file name in the sidecar definition yaml matches the file name specified in the variable definition. That is, .spec.volumes.configMap.items.path(file name) = "my-config.conf" • The mount path in the sidecar definition yaml matches the mount path

Environment variable name	Description
	specified in the variable value. That is, .spec.container.volumeMounts.mountPath(sidecar container mount path) ="/tmp/configmaps/data" If you use an absolute path, you can mount a ConfigMap file inside the sidecar container at any location. **As a relative path:** For example: If you set the variable as: VCS_CONFIGMAP = "custom_file_name.conf", You must ensure that: • The file name in the sidecar definition yaml matches the file name in the variable definition. That is, .spec.volumes.configMap.items.path(file name) = "custome_file_name.conf" • The mount path in the sidecar definition yaml matches the default mount path of the variable. That is, .spec.container.volumeMounts.mountPath(side-car container mount path) = /var/VRTSvcs/container/configmap **Not setting up the variable:** For example: If you do not set the VCS_CONFIGMAP variable, then you must ensure that the file name and mount path specified in the sidecar definition yaml matches the default file name and the default mount path. That is, .spec.volumes.configMap.items.path(file name) = "config-file.conf" .spec.container.volumeMounts.mountPath(side-car container mount path) = "/var/VRTSvcs/container/configmap"
VCS_CUSTOMSCRIPT	Use this variable to customize the sidecar container with parameters for VCS configurations. You can specify HA commands in this custom script to configure VCS parameters such as ToleranceLimit, MonitorTimeout, MonitorInterval, and OfflineMonitorInterval. For more information on configuring VCS parameters, refer to the Cluster Server Administrator's Guide. **Note**: You must place the custom script only at a volume shared between the sidecar container and the host and provide absolute path for the custom script.

Section 5: Monitoring applications within containers

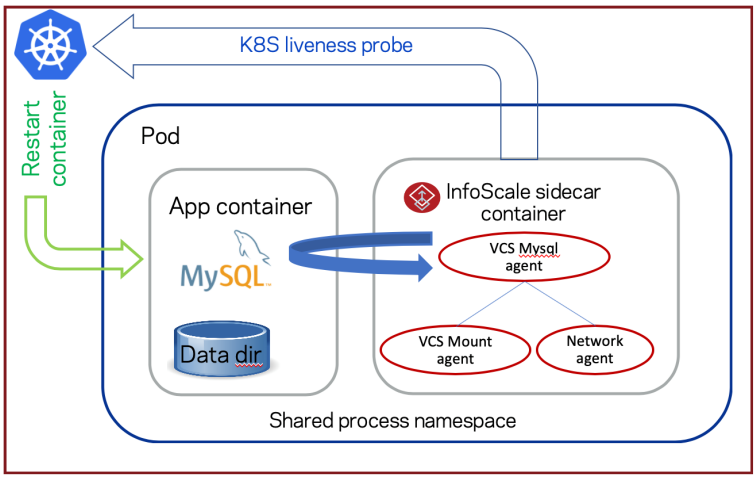
Monitoring applications in a InfoScale sidecar-based deployment

Applications and services often require monitoring to ensure uninterrupted operations. If the peripheral tasks like monitoring and logging are tightly coupled into the application, an outage in one of these application components can affect the entire application.

To address this challenge, Kubernetes provides the sidecar component that allows you to implement such peripheral tasks as separate components or services but maintain close interdependence on each other.

In a typical InfoScale setup in Kubernetes environment, you can use a sidecar container to augment and improve the efficiency of an application container without disturbing it. Sidecar containers are scheduled on the same host along with the application container and share several resources such as disks, network, filesystem, hostname and many other namespaces.

Note: InfoScale sidecar-based deployment only supports MySQL 5.x or 8.x along with the custom applications in the same application container.



Thus, a sidecar is a utility container in the pod that supports the main container. It can be considered as a library functionality for the containers.

Along with the InfoScale agents, InfoScale leverages the container probing feature of Kubernetes to monitor such applications in a sidecar-based deployment.

Note: With InfoScale 7.4.3 in a Kubernetes environment, only one application container from a pod can be monitored using InfoScale sidecar container.

The InfoScale VCS agents performs the following application monitoring operations for the application container:

- **Processes monitoring**: Monitors the application processes and ensures that all the important application processes are running.

  For example, in case of MySQL container, process monitoring ensures that the 'mysqld' process is running.
- **In-depth monitoring**: Monitors the application by simulating it and using the health check commands and the sample database queries.

  For example, in Oracle, in-depth monitoring ensures that the database is responsive by executing a sample database insert query using the SqlTest.pl script.
- **Application storage dependency monitoring**: Monitors the Mount resources by using the InfoScale agents to ensure storage availability within the container.

- **Application network dependency monitoring**: Monitors the NIC resources by using the InfoScale agents to ensure network availability within the container.

Probes for monitoring container health

Probes are defined at each container level, that is for each application container as well as sidecar container. Kubernetes use the probes to determine the state of each resource in that respective container. InfoScale uses the following types of probes in a containerized environment:

Liveness probe

Liveliness probe enables the kubelet to detect when an application is in a bad state. It also instructs the kubelet to restarts the pod to restore availability of that an unhealthy container if it fails the liveness check. Liveness probes can help the application to recover from a deadlock situation.

The liveliness probe provides results as Success, Failure, and Unknown.

If a probe is not defined, the default is Success.

During the liveness probe, InfoScale agents perform different levels of monitoring for the applications such as process monitoring, in-depth monitoring, and storage and network dependency monitoring.

Startup probe

Startup probes are executed at startup, that is when the application is started or fully initialized. These probes must be optimized to accommodate slow starting containers or applications with unpredictable initialization processes. This optimization ensures that such containers do not get killed by the kubelet before they are even fully initialized.

To configure a startup probes to handle the worst-case scenario, we can use the **failureThreshold** and **periodSeconds** parameters in the yaml file to address the unpredictable initialization time.

For example, set **failureThreshold** to 5 and **periodSeconds** to 60 if you want Kubernetes to wait for 5 x 60 = 300s (5 min) for the container to fully initialize before it fails.

The startup probe also disables the liveness checks until it succeeds.

Veritas recommends that you configure startup probe for InfoScale sidecar containers in such a way that:

- The probe allows enough time for the VCS engine to get fully initialized (RUNNING state).
- The probe allows enough time for the MySQL database to fully start and accounts for some extra time that may be required in database recovery.

Configuring probes

You can configure a liveness probe by updating the pod definition yaml file while creating a container. That is, while creating a pod you must update the pod definition yaml and add the probe definition to it.

InfoScale provides the following probe scripts for using with the sidecar and the application container. These scripts are available as a part of the VRTScontainer package. When you install the VRTScontainer package, these scripts are available at the following location:

- Probe script for Sidecar container:
  `/var/VRTScontainer/app_status/probes/MySQL/MySQL_InfoScale_sidecar_container_Liveness_probe.pl`
- Probe script for MySQL application container:
  `/var/VRTScontainer/app_status/probes/MySQL/MySQL_application_Liveness_probe.sh`

These scripts can be used to configure liveliness as well as startup probes.

To configure a probe, perform the following steps:

1. Ensure that the **VRTScontainer** package is installed on every InfoScale cluster node.
   You can download the appropriate platform package from https://sort.veritas.com/agents/.
2. Mount the path `/var/VRTScontainer/app_status` by using the hostPath method.
   Ensure that the path is mounted at the same `/var/VRTScontainer/app_status` location inside the InfoScale sidecar container as well as the MySQL application container.
3. Update the pod definition in the yaml by adding one of the following commands along with the other parameters.
   - To configure a probe for a Sidecar container, use:
     /var/VRTScontainer/app_status/probes/MySQL/MySQL_InfoScale_sidecar_container_Liveness_probe.pl
   - To configure a probe for an application container, use:
     /var/VRTScontainer/app_status/probes/MySQL/MySQL_application_Liveness_probe.sh

The following yaml file snippet shows the sample liveness probe definition included in a pod definition:

```
livenessProbe:
      exec:
        command:
        -
/var/VRTScontainer/app_status/probes/MySQL/MySQL_application_Liveness_probe.sh
        # Full file path shared between application container and sidecar
      initialDelaySeconds: 180
      periodSeconds: 30
      timeoutSeconds: 60
      successThreshold: 1
      failureThreshold: 1
```

The following yaml file snippet shows the sample startup probe definition included in a pod definition:

```
startupProbe:
      exec:
        command:
        -
/var/VRTScontainer/app_status/probes/MySQL/MySQL_application_Liveness_probe.sh
        # Full file path shared between application container and sidecar
      initialDelaySeconds: 180
      periodSeconds: 30
      timeoutSeconds: 60
      successThreshold: 1
      failureThreshold: 6
```

How does monitoring work in a sidecar-based deployment?

For the InfoScale agents to perform any monitoring operations, it is necessary that VCS is operational. When a sidecar container starts, it performs the following operations to ensure proper monitoring:

1. Sidecar container start VCS onenode within container using the /opt/VRTSvcs/bin/setup_availability_container.sh utility.
2. It verifies that VCS is running in the InfoScale sidecar and validated the license.
3. If VCS is not running in the sidecar, the sidecar container restarts itself.
4. Based on the ConfigMap and secret details, the sidecar container creates the VCS service group and resources for monitoring. It also adds the resource dependencies and the MySQL users.
5. Starts VCS so that the InfoScale agents perform the application and resource monitoring.
6. Executes any addition VCS customization scripts, if specified in the sidecar deployment yaml.

   The path to this custom script is specified as an environment variable VCS_CUSTOMSCRIPT in InfoScale sidecar container.
7. The liveness probe script checks the state of each VCS resources monitored by the agents.
8. If all the VCS resources are online, the liveness probe script creates a service group status file in the `/var/VRTScontainer/app_status` directory for each pod. The name of this file is in the `<container_hostname>.<ServiceGroupName>.ONLINE` format.

   For example: `/var/VRTScontainer/app_status/mysqlpod.mysql_grp.ONLINE`
9. If any of the VCS resources is in offline, partial, or faulted state, the liveness probe script either does not create this service group status file or deletes the service group status file if it exists.
10. Based on the availability of this file, the liveliness probe of the application container decides if the container continues to operate or must be restarted.

    That is, if the application container finds a `<container_hostname>.<ServiceGroupName>.ONLINE` file, the container continues to operate as normal. If this file is not available, the probe script restarts the container.

Parameters to configure and control the behavior of probes

All the probe options contain the following parameters. The values assigned to these parameters define the behavior of the probe.

Table 13: Typical parameters recommended by InfoScale VCS to control the behavior of probes

Parameter	Description
initialDelaySeconds	Specifies the time in seconds for which the kubelet must wait before initiating the first probe after creating the container. Recommended value for InfoScale VCS: 180 seconds
periodSeconds	Specifies the time in seconds for which the kubelet must wait before initiating a new probe. Recommended value for InfoScale VCS: 30 seconds
TimeoutSeconds	Specifies the time in seconds after which the probe times out. Recommended value for InfoScale VCS: 60 seconds

Parameter	Description
successThreshold	Specifies the minimum number of consecutive successes for the probe to be considered successful after a probe fails. Recommended value for InfoScale VCS: 1
failureThreshold	Specifies the number of times that the probe can fail before the probe marks the container as unhealthy and restarts it. Recommended value for InfoScale VCS: 2.

Setting up an InfoScale sidecar container

Prerequisites

- All nodes have the same time. If you are not running the Network Time Protocol (NTP) daemon, make sure the time on all the systems comprising your cluster is synchronized.
- InfoScale is Installed on the host where the pod is running with security on.
- The VRTScontainer package is installed on every InfoScale cluster node.

  You can download the package based on the host operating system version from one the following locations:

  - RHEL 7: https://sort.veritas.com/agents/detail/20314
  - RHEL 8: https://sort.veritas.com/agents/detail/20315
  - SLES 12: https://sort.veritas.com/agents/detail/20316
  - SLES 15: https://sort.veritas.com/agents/detail/20317
- The InfoScale sidecar image is loaded in the image repository using the container runtime specific command.
- The value of the VCS_RUN_IN_SIDECAR_CONTAINER environment variable is set to 1.
- The Mount agent are configured to run in OnOnly Mode.
- The VCS AUTHSERVER certificates are mounted within sidecar container. You can use the hostPath method to mount the certificates.
- The Cluster UUID is specified in the sidecar container.
- The `/var/VRTScontainer/app_status` directory is mounted within the sidecar container and the application container.
- The required ConfigMaps and Secrets, probe scripts, and the REST Server details are made available within the sidecar container.

  Note: Veritas recommends to also encrypt the Kubernetes secrets using EncryptionConfiguration.
- Process name sharing is enabled between the containers.

To set up the sidecar container:

1. Update the yaml for container setup with the obtained information.

   For example, update the yaml as shown in the following sample:

```
volumeMounts:
        - mountPath: /var/VRTSvcs/container/AUTHSERVER
```

```
        name: vcsauthserver-certstore
        readOnly: true
      - mountPath: /var/VRTSvcs/container/uuid
        name: clusuuid
        readOnly: true
      - mountPath: /var/VRTScontainer/app_status
        name: appstatus
      - mountPath: /var/VRTSvcs/container/configmap/
        name: mapvol
        readOnly: true
      - name: secret-vol
        mountPath: /var/VRTSvcs/container/secret
        readOnly: true
volumes:
 - name: vcsauthserver-certstore
   hostPath:
     path: /var/VRTSvcs/vcsauth/bkup/VCS_SERVICES
     type: File
 - name: clusuuid
   hostPath:
     path: /etc/vx/.uuids/clusuuid
     type: File
 - name: appstatus
   hostPath:
     path: /var/VRTScontainer/app_status
     type: Directory
 - name: mapvol
   configMap:
     name: mysql-configmap
 - name: secret-vol
   secret:
     secretName: mysql-secret
     defaultMode: 0400
     items:
     - key: mysqlpass
       path: MySQLAdminPasswd
 - name: mysqldatadir
   persistentVolumeClaim:
     claimName: csi-infoscale-pvc
```

2. Run the following command to setup and start the container.

```
# kubectl -f create <name.yaml>
```

Configuring the licensing feature within a sidecar container

In order to reduce the licensing overhead inside an application or a sidecar container, Veritas uses the InfoScale REST server to validates the licenses. This method avoids the need not install license keys in every application or sidecar container.

You must setup an InfoScale REST server with the appropriate licensing skew. To validate an InfoScale container, the InfoScale REST server must have the InfoScale Enterprise license installed.

When VCS is started inside a container, VCS queries the REST server for the required license. Cluster formation and configuration proceeds only if the REST server validates and responds with the license details.

To enable configuring the licensing feature within a container, you must set the address and port of the REST server. You can do this in one of the following ways:

- Set the VX_REST_URL environment variable that points to the InfoScale REST server.
  For example: export VX_REST_URL=https://11.22.33.44:1234
- Specify the VX_REST_URL environment variable in the yaml file while configuring a container.

You must set this variable before starting and configuring VCS. In absence of this environment variable, the license validation fails and prevents the VCS from starting.

The setup_availability_container.sh utility

To ensure proper container creation, VCS configuration, and application monitoring, Veritas provides the **setup_availability_container.sh** utility. This utility is available at `${VCSHOME}/bin/setup_availability_container.sh` location.

To use this utility, you must update your container definition by adding the path of this utility as an argument.

For example, in the container definition, you can define:

```
image: <InfoScale_vcs_image_from_local_repository>
command: [ "/bin/bash", "-c", "--" ]
args: [ "/opt/VRTSvcs/bin/setup_availability_container.sh; while true; do sleep 30;
done;" ]
 securityContext:
```

veritas provides a sample container setup script at `${VCSHOME}/bin/ApplicationContainer_setup_apache_container.sh`.

You can also create a customized script for your container setup and call the **setup_availability_container.sh** utility from it to create a container and enable monitoring.

The **setup_availability_container.sh** utility take care of the following operations:

Table 14: Typical operations performed by the setup_availability_container.sh utility

Operations	Description
Install necessary packages for InfoScale Availability container	Installs packages such as: httpd, hostname, ksh, net-tools, iproute, VRTSperl, VRTSvcsag, VRTSvcs, VRTSvlic, VRTSvcsea, VRTSacclib, VRTSmysql (if monitoring MySQL), and so on

Operations	Description
	Note: If the deployed InfoScale image contain these executables, you may edit the utility and comment out the install_rpms() call.
Configure cluster UUID	InfoScale cluster UUID (Universally Unique ID) uniquely identifies a cluster and prevents system nodes from different clusters to accidently join an VCS clusters that they do not belong to. **Note**: If the UUID of an InfoScale REST server is already mounted, you may edit the utility and comment out the configure_uuid() call.
Configure secure cluster	By default, VCS inside an application or sidecar container is configured in secure mode. This utility: • Creates and updates the EAT file with necessary permission • Starts the VCS AUTH sever in FIPS mode based on the host FIPS mode. That is, if the FIPS mode is enabled on the host, the VCS AUTH server starts with FIPS=1 else it starts with FIPS=0 • Update the cluster configuration file (main.cf) • Starts VCS cluster in secure mode **Note**: As Secure mode is default and recommended, configure_secure_cluster() is a mandatory function of the utility.
Configure licensing within a container	Configures the REST server for licensing.
Ensuring secure communication between InfoScale containers and the InfoScale REST server	This utility ensures secure communication between the InfoScale container and the InfoScale REST server by performing the following operations: • Decrypting the private keys • Obtaining the broker certificate • Obtaining the HAD certificate • Obtaining the tokens from VX_REST_URL • Authorizing with InfoScale REST server at VX_REST_URL This utility also ensures that InfoScale REST server certificates, public keys, and cluster UUID file are mounted inside container.
Configure VCS	The create_vcs_config function in this utility configures VCS within the container. **Note**: To create VCS configuration, you may choose to use the existing configuration or update the configuration. To update the configuration, you may update the create_vcs_coinfig function in the utility or create a separate ${VCSHOME}/bin/create_vcs_config.pl file with the required VCS configurations.

Sample configuration

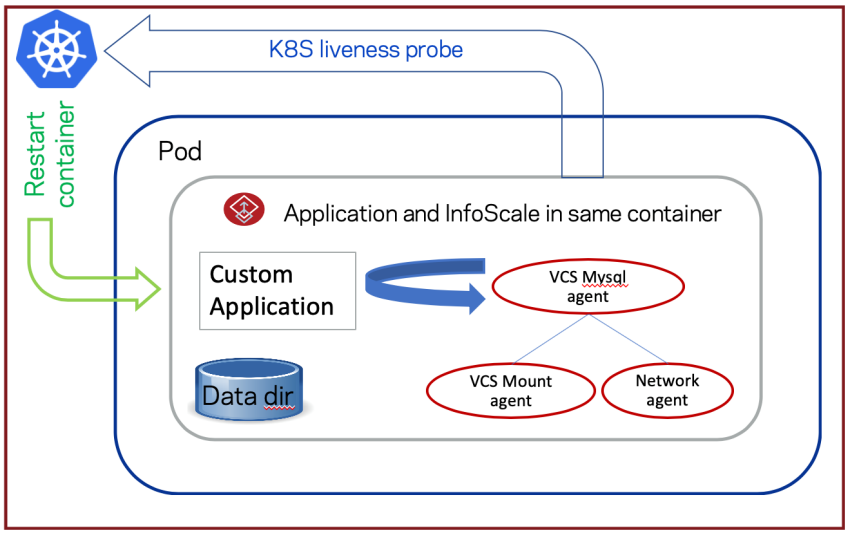
```
apiVersion: v1
kind: Pod
metadata:
  name: mysqlpod
spec:
  shareProcessNamespace: true
  containers:
  - name: mysql-test
    image: mysql
    envFrom:
    - secretRef:
        name: mysql-secret
    livenessProbe:
      exec:
        command:
        -
/var/VRTScontainer/app_status/probes/MySQL/MySQL_application_Liveness_probe.sh
        # Full file path shared between application container and sidecar
      initialDelaySeconds: 180
      periodSeconds: 30
      timeoutSeconds: 60
      successThreshold: 1
      failureThreshold: 1
    startupProbe:
      exec:
        command:
        -
/var/VRTScontainer/app_status/probes/MySQL/MySQL_application_Liveness_probe.sh
        # Full file path shared between application container and sidecar
      initialDelaySeconds: 180
      periodSeconds: 30
      timeoutSeconds: 60
      successThreshold: 1
      failureThreshold: 6
    volumeMounts:
        - mountPath: /var/VRTScontainer/app_status
          name: appstatus
        - mountPath: "/var/lib/mysql"
          name: mysqldatadir
  - name: mysql-monitor
    env:
    - name: VX_REST_URL
      value: https://10.209.57.200:5636
    - name: VCS_CONFIGMAP
      value: /var/VRTSvcs/container/configmap/config-file.conf
    - name: VCS_SECRET_MySQLAdminPasswd
```

```
      value: /var/VRTSvcs/container/secret/MySQLAdminPasswd
    # <specify InfoScale vcsimage from local repository>
    #image: 10.209.57.62:5000/vcsimage:v1
    image: 10.209.57.62:5000/ubi7:private
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "/opt/VRTSvcs/bin/setup_availability_container.sh; while true; do sleep
30; done;" ]
    securityContext:
      capabilities:
        add:
        - SYS_PTRACE
    stdin: true
    tty: true
    livenessProbe:
      exec:
        command:
        -
/var/VRTScontainer/app_status/probes/MySQL/MySQL_InfoScale_sidecar_container_Livene
ss_probe.pl
      initialDelaySeconds: 180
      periodSeconds: 30
      timeoutSeconds: 30
      successThreshold: 1
      failureThreshold: 2
    volumeMounts:
        - mountPath: /var/VRTSvcs/container/AUTHSERVER
          name: vcsauthserver-certstore
          readOnly: true
        - mountPath: /var/VRTSvcs/container/uuid
          name: clusuuid
          readOnly: true
        - mountPath: /var/VRTScontainer/app_status
          name: appstatus
        - mountPath: /var/VRTSvcs/container/configmap/
          name: mapvol
          readOnly: true
        - name: secret-vol
          mountPath: /var/VRTSvcs/container/secret
          readOnly: true
  volumes:
   - name: vcsauthserver-certstore
     hostPath:
       path: /var/VRTSvcs/vcsauth/bkup/VCS_SERVICES
       type: File
   - name: clusuuid
     hostPath:
       path: /etc/vx/.uuids/clusuuid
       type: File
```

```
- name: appstatus
  hostPath:
    path: /var/VRTScontainer/app_status
    type: Directory
- name: mapvol
  configMap:
    name: mysql-configmap
- name: secret-vol
  secret:
    secretName: mysql-secret
    defaultMode: 0400
    items:
    - key: mysqlpass
      path: MySQLAdminPasswd
- name: mysqldatadir
  persistentVolumeClaim:
    claimName: csi-infoscale-pvc
```

Custom application monitoring within application container

InfoScale containers can monitor application even without having a sidecar container. In this case, you can install VCS in the application container. Thus, the InfoScale packages and the managed application resides and operate from the same container.



When InfoScale is configured to run in the application container, you must ensure the following:

- Install InfoScale within the application container
- Install the commands and utilities like ps, cat, ifconfig, and so on in the application container to ensure proper monitoring my VCS agents
- Create the required VCS resources for monitoring

- Set the value of the VCS_RUN_IN_APP_CONTAINER environment variable to 1
- Define the liveness probe for your application container to create a service group status file and reflect the service group state.

  For example:
  - The liveness probe script creates a service group status file in the directory that is specified in the yaml for each pod only if all the VCS resources are online.
  - The liveness probe script either does not create this service group status file or deletes the service group status file if it exists if any of the VCS resources is in offline, partial, or faulted state.

To ensure proper container creation, VCS configuration, and application monitoring, Veritas provides the setup_availability_container.sh utility. This utility is available at `${VCSHOME}/bin/setup_availability_container.sh`.

To use this utility, you must update your container definition by adding the path of this utility as an argument.

As a sample, Veritas provides a script to sets up the Apache application container that permits monitoring operations. The script is located at the following location:

`${VCSHOME}/bin/ApplicationContainer_setup_apache_container.sh`

This setup script internally invokes the setup_availability_container.sh utility.

You can also create a customized script for your container setup and call the setup_availability_container.sh utility from it to create a container and enable monitoring.

Once the application containers are configured, the InfoScale agents and Kubernetes probe features starts the monitoring operations the same way as in a sidecar-based deployment.

Sample configurations

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apachepod-deployment
  labels:
    app: apachepod-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apachepod-deployment
  template:
    metadata:
      labels:
        app: apachepod-deployment
    spec:
      containers:
      - name: apache-monitor
        env:
         - name: VX_REST_URL
           value: https://10.209.69.143:443
        image: 10.209.57.62:5000/apachevcs
```

```
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "/custom_app/setup_apache.sh; while true; do sleep 30; done;" ]
    ports:
    - containerPort: 80
      hostPort: 81
    securityContext:
      capabilities:
        add:
        - SYS_PTRACE
    stdin: true
    tty: true
    livenessProbe:
      exec:
        command:
        - /custom_app/liveness_sidecar.pl
      initialDelaySeconds: 120
      periodSeconds: 30
      timeoutSeconds: 60
      successThreshold: 1
      failureThreshold: 1
    volumeMounts:
        - mountPath: /custom_app
          name: appstatus
        - mountPath: /var/VRTSvcs/container/AUTHSERVER
          name: vcsauthserver-certstore
          readOnly: true
        - mountPath: /etc/vx/.uuids/clusuuid
          name: clusuuid
          readOnly: true
  volumes:
   - name: appstatus
     hostPath:
       path: /custom_app
       type: Directory
   - name: vcsauthserver-certstore
     hostPath:
       path: /var/VRTSvcs/vcsauth/bkup/VCS_SERVICES
       type: File
   - name: clusuuid
     hostPath:
       path: /etc/vx/.uuids/clusuuid
       type: File
```

Section 6: Troubleshooting

Enabling debug logs for a containerized InfoScale deployment

VCS generates two types of logs: the engine log and the agent log. Log file names are appended by letters; letter A indicates the first log file, B the second, and C the third. After three files, the least recent file is removed, and another file is created. For example, if engine_A.log is filled to its capacity, it is renamed as engine_B.log and the engine_B.log is renamed as engine_C.log. In this example, the least recent file is engine_C.log and therefore it is removed. You can update the size of the log file using the LogSize cluster level attribute.

The VCS logs are located in the `/var/VRTSvcs/log/` directory.

In case of a failure in an InfoScale in sidecar container, the pod or the sidecar container gets restarted. During the restart, the VCS log files gets deleted and consequently limits the possibility of tracking the previous failures in the sidecar container. Therefore, it is important to preserve the logs for any previous failures so that you can identify the cause of the failures and fixed them appropriately. You can ensure availability of such logs even after the restart with the help of Persistent storage.

To make such logs persistently available, you must mount the persistent volume at the `/var/VRTSvcs/log` directory within a sidecar container.

For example, to create a persistent volume and enable logging, perform the following steps:

1. Create persistent volume and in the hostPath attribute specify the path on the host where you want to store the logs.
   `# kubectl create -f pv-volume.yaml`

```
---
apiVersion: v1
kind: PersistentVolumes
metadata:
  name: pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWritemany
  hostPath
    path: "/mnt/pod_name"
```

2. Create a Persistent Volume Claim .
   `# kubectl create -f pv-claim.yaml`

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: task-pv-claim1
spec:
   storageClassName: manual
   accessModes:
       - ReadWriteMany
resources:
```

```
requests:
storage: 1Gi
```

3. Configure the pod to use the PVC in the sidecar container.

```
# kubectl create -f mysqlpod.yaml
```

```
apiVersion:v1
kind: Pod
metadata:
   name: mysqlpod
spec:
   containers:
     -name: sidecar-monitor
      image: vcsimage:latest
      command: [ "/bin/bash", "-c", "--"sss]
      args: ["/opt/VRTSvcs/bin/setup_availability_container.sh; while true; do
sleep 30; done;" ]
      volumeMounts:
          - mountPath: /var/VRTSvcs/log
            name: pv-storage
      volumes:
         - name: pv-storage
           persistentVolumeClaim:
           claimName: pv-claim
```

Note that:

- It is optional to use a PVC to collect log data for analysis.
- If you plan to use a PVC for collecting failure logs, you must use one PVC per pod.
- You can define any location on the host to store the logs. However, in such case, you must ensure that the same location is not used by multiple pods, else the log files may get overwritten by different pods.

Evacuating the node which has CSI controller pod running

In case of voluntary and involuntary disruptions, you must safely evacuate the pods from a node before you perform maintenance on that node. Safe evacuation allows the containers to gracefully terminate and will respect the specified PodDisruptionBudgets. You can use the **kubectl drain** command to safely evacuate the pods.

However, as the CSI controller pod uses empty directory for socket communications, the statefulset applications are not moved during the evacuation operation, and the drain command fails with the following error: 'cannot delete Pods with local storage'

In such case, you must run the **kubectl drain** command with the **--delete-local-data** option to evacuate the pods from current node and rescheduled them on another node.

For example:

```
# kubectl drain <node_name> --force --ignore-daemonsets --delete-local-data
```

Recovering from a disabled file system

A file system usually becomes disabled because of disk errors. Disk failures that disable a file system should be fixed as quickly as possible.

In a container environment, if the file system is disabled, it may cause the PVCs to become inaccessible for the pods. In such case, each mount must be unmounted and remounted again.

To do so, you can stop all the pods and restart them so that the file system is mounted again.

Before, you take any action, you must ensure that there are no underlying storage issues before recovering a disabled file system or a file system that requires a full file system check. You can use the **fsck** command.

For information on troubleshooting disks, see the chapter on recovering from hardware failure in the *Veritas InfoScale Troubleshooting Guide - Linux*.

Before you repair the VxFS file system, Veritas recommends that you check it first to evaluate the possible structural damage. In a Storage Foundation and High Availability (SFHA) Solutions environment, you should freeze the service group that contains the mount point in trouble before you do any manual recovery procedures.

Warning: You can use **fsck** to check and repair a VxFS file system; however, improper use of the command can result in data loss. Do not use this command unless you thoroughly understand the implications of running it. If you have any questions about this command, contact Symantec Technical Support.

You can check the structure of the file system in one of the following ways:

- Run the **fsck** command with **-n**, **full**, and **nolog** options. The -n option ensures that all fsck prompts are answered "no" and the file system is not opened for writing:
  ```
  # fsck -V vxfs -n -o full,nolog /dev/vx/rdsk/diskgroup/volume
  ```
- Use the metasave script for your operating system platform to generate a copy of the metadata of the file system, then replay the metasave and run **fsck** with the **-y** option command to evaluate possible structural damage.

Warning: If you have any questions about these procedures or do not fully understand the implications of the fsck command, contact Technical Support.

For more information on the fsck command, see the **fsck_vxfs(1M)** manual page.

Appendix: YAML files

Appendix A: StorageClass YAMLs

The StorageClass yaml files contains parameters like:

Attributes	Description
fsType	Specifies the supported filesystem. InfoScale supports only VxFS file system for Persistent Volumes.
layout	Specifies the InfoScale volume layout. Supported layouts: stripe, mirror, stripe-mirror, mirror-stripe, concat, concat-mirror, mirror-concat. This parameter is optional.
faultTolerance	Specifies the number of disk or host failures a storage object can tolerate. This parameter is optional.
nstripe	Specifies the number of stripe columns to use when creating a striped volume. This parameter is optional.
stripeUnit	Specifies the stripe unit size to use for a striped volume. This parameter is optional.
mediatype	Specifies disks with the specified media type. All disks with the given mediatype are selected for volume creation. Supported values: hdd, ssd This parameter is optional.
encryption	Specifies whether to store encrypted data on disks or not. This parameter is optional

csi-infoscale-performance-sc

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-infoscale-performance-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
```

```
provisioner: org.veritas.infoscale
reclaimPolicy: Delete
allowVolumeExpansion: true
parameters:
  fstype: vxfs

  # (optional) Specifies a volume layout type.
  # Supported layouts: stripe, mirror, stripe-mirror, mirror-stripe, concat,
concat-mirror, mirror-concat
  layout: "stripe-mirror"
  # (optional) Specifies the number of disk or host failures a storage object can
tolerate.
  faultTolerance: "1"
  # (optional) Specifies disks with the specified media type. All disks with the
given mediatype are selected for volume creation.
  # Supported values: hdd, ssd
  mediaType: "ssd"
```

csi-infoscale-sc

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

  # (optional) Specifies a volume layout type.
  # Supported layouts: stripe, mirror, stripe-mirror, mirror-stripe, concat,
concat-mirror, mirror-concat
  # If omitted, InfoScale internally chooses the best suited layout based on the
environment.
  # layout: "mirror"
  # (optional) Specifies the number of disk or host failures a storage object can
tolerate.
  # faultTolerance: "1"
  # (optional) Specifies the number of stripe columns to use when creating a
striped volume.
  # nstripe: "3"
  # (optional) Specifies the stripe unit size to use for striped volume.
  # stripeUnit: "64k"
```

```
  # (optional) Specifies disks with the specified media type. All disks with the
given mediatype are selected for volume creation.
  # Supported values: hdd, ssd
  # mediaType: "hdd"
  # (optional) Specifies whether to store encrypted data on disks or not.
  # Valid values are true or false
  # encryption: "false"
```

csi-infoscale-resiliency-sc.yaml

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-infoscale-resiliency-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: org.veritas.infoscale
reclaimPolicy: Delete
allowVolumeExpansion: true
parameters:
  fstype: vxfs

  # (optional) Specifies a volume layout type.
  # Supported layouts: stripe, mirror, stripe-mirror, mirror-stripe, concat,
concat-mirror, mirror-concat
  layout: "mirror"
  # (optional) Specifies the number of disk or host failures a storage object can
tolerate.
  faultTolerance: "2"
```

csi-infoscale-secure-sc.yaml

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-infoscale-secure-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: org.veritas.infoscale
reclaimPolicy: Delete
allowVolumeExpansion: true
parameters:
  fstype: vxfs
```

```
  # (optional) Specifies a volume layout type.
  # Supported layouts: stripe, mirror, stripe-mirror, mirror-stripe, concat,
concat-mirror, mirror-concat
  layout: "mirror"

  # (optional) Specifies the number of disk or host failures a storage object can
tolerate.
  faultTolerance: "1"

  # (optional) Specifies whether to store encrypted data on disks or not.
  # Valid values are true or false
  encryption: "true"
```

Appendix B: SnapshotClass YAMLs

csi-infoscale-snapclass.yaml

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: csi-infoscale-snapclass
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true"
driver: org.veritas.infoscale
deletionPolicy: Delete
#parameters:
  # (optional) Specifies the type of the snapshot to be created.
  # If omitted, InfoScale by default creates space-optimized snapshot.
  # Supported values: space-optimized, full-instant
  # snapType: space-optimized

  # (optional) Specifies the size of the cache volume to be created for space-
optimized snapshots.
  # If omitted, InfoScale internally chooses the cacheSize as 30% of orignial
volume size.
  # cacheSize: 500m
```

The SnapshotClass YAML files contains parameters like:

Attributes	Description
snapType	Specifies the type of the snapshot to be created. If you do not specify this value, InfoScale by default creates a space-optimized snapshot. Supported values: space-optimized, full-instant. This parameter is optional.
cacheSize	Specifies the size of the cache volume to be created for space-optimized snapshots. If you do not specify this value, InfoScale internally calculates the cacheSize as 30% of the original volume size. This parameter is optional.

Appendix C: dynamic_provisioning

csi-dynamic-pvc.yaml

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-infoscale-pvc
spec:
  storageClassName: csi-infoscale-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

Attributes	Description
storageClassName	Specifies the name of the previously created InfoScale StorageClass.
accessModes	Specifies the access mode. Supported values: ReadWriteOnce, ReadOnlyMany, ReadWriteMany.

csi-dynamic-snapshot.yaml

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
```

```
metadata:
  name: csi-dynamic-snapshot
spec:
  volumeSnapshotClassName: csi-infoscale-snapclass
  source:
    persistentVolumeClaimName: csi-infoscale-pvc
```

Attributes	Description
volumeSnapshotClassName	Specifies the name of the previously created InfoScale volumeSnapshotClass.
persistentVolumeClaimName	Specifies the name of the PVC from which a snapshot should be created.

csi-dynamic-snapshot-restore.yaml

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-infoscale-snapshot-restore
spec:
  storageClassName: csi-infoscale-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  dataSource:
    name: csi-dynamic-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

Attributes	Description
storageClassName	Specifies the name of the previously created InfoScale storageClass . This storageClass should be same as that of the source PVC.
storage	Size of the PVC that will get restored from snapshot. This value should be same as that of snapshot size.

csi-dynamic-volume-clone.yaml

```
---
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-infoscale-volume-clone
spec:
  storageClassName: csi-infoscale-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: csi-infoscale-pvc
```

Attributes	Description
storageClassName	Specifies the name of the previously created InfoScale storageClass. This storageClass should be same as that of the source PVC.
accessModes	Specifies the supported accessModes. Supported values: ReadWriteOnce, ReadOnlyMany, ReadWriteMany
storage	Specifies the size of the clone PVC. This value should be same as that of the source PVC.

Appendix D: static_provisioning

csi-static-pv.yaml

```
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: csi-infoscale-pv
  annotations:
    pv.kubernetes.io/provisioned-by: org.veritas.infoscale
spec:
  storageClassName: csi-infoscale-sc
  persistentVolumeReclaimPolicy: Delete
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  csi:
    driver: org.veritas.infoscale
```

```
    # Please provide pre-provisioned Infoscale volume name.
    volumeHandle: testVol
    fsType: vxfs
```

Attributes	Description
storageClassName	Specifies the name of the previously created InfoScale storageClass.
storage	Specifies the size of the PV. This value should be similar to that of VxVM volume mentioned in volumeHandle.
accessModes	Specifies the accessModes. Supported values: ReadWriteOnce, ReadOnlyMany, ReadWriteMany
driver	Specifies the InfoScale CSI driver 'org.veritas.infoscale'.
volumeHandle	Specifies the name of the existing VxVM Volume.
fsType	Veritas File System (vxfs)

csi-static-pvc.yaml

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

Attributes	Description
accessModes	Specifies the accessModes. Supported values: ReadWriteOnce, ReadOnlyMany, ReadWriteMany.
storageClassName	Specifies the name of the previously created InfoScale storageClass.

csi-static-snapshot-content.yaml

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotContent
metadata:
  name: csi-static-snapshot-content
spec:
  deletionPolicy: Retain
  driver: org.veritas.infoscale
  source:
    # Provide pre-provisioned Infoscale snapshot volume name
    snapshotHandle: testSnapVol
  volumeSnapshotRef:
    name: csi-static-snapshot
    namespace: default
```

Attributes	Description
driver	Specifies the InfoScale CSI driver 'org.veritas.infoscale'.
snapshotHandle	Specifies the name of the InfoScale snapshot volume that is already created on the host.
volumeSnapshotRef	Specifies the name of the VolumeSnapshot object.

csi-static-snapshot.yaml

```
---
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: csi-static-snapshot
spec:
  volumeSnapshotClassName: csi-infoscale-snapclass
  source:
    volumeSnapshotContentName: csi-static-snapshot-content
```

Attributes	Description
volumeSnapshotClassName	Specifies the name of the previously created InfoScale volumeSnapshotClass.
volumeSnapshotContentName	Specifies the name of the previously created volumeSnapshotContentName.

csi-static-snapshot-restore.yaml

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-static-snapshot-restore
spec:
  storageClassName: csi-infoscale-sc
  dataSource:
    name: csi-static-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

Attributes	Description
storageClassName	Specifies the name of the previously created InfoScale storageClass . This storageClass should be same as that of the source PVC.
storage	Specifies the size of the PVC that will get restored from the snapshot. This value should be same as the snapshot size.

Appendix E: Sample application YAML

csi-pod.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    name: redis
spec:
  containers:
    - name: redis
      image: redis
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - mountPath: "/mnt/veritas"
          name: vol1
```

```
volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: csi-infoscale-pvc
```

Attributes	Description
persistentVolumeClaim	Specifies the name of the InfoScale PVC that you want to use for the application container.