



Openshift Plugin

Bacula Systems Documentation

Contents

1	OpenShift: Features	4
2	OpenShift: Installation	5
2.1	Prerequisites	5
2.2	Installation with Package Manager	5
2.3	bacula-backup Proxy Pod Image Installation	6
2.4	Advanced Bacula Backup Proxy Pod Deployment	6
2.5	Longhorn Snapshot Installation	9
3	OpenShift: Configuration	10
3.1	Generic Plugin Parameters	10
3.2	Estimate and Backup Plugin Parameters	11
3.3	Backup and Restore Plugin Parameters	18
3.4	Restore Plugin Parameters	19
3.5	Tuning Plugin Parameters	20
3.6	Fileset Examples	21
4	OpenShift: Operations	23
4.1	Backup	23
4.2	Restore	34
4.3	Resource Listing	44
5	OpenShift: Autonomous Actions	46
5.1	Auto-Skip of PVCs	46
5.2	Auto-Detection of Filesystem Format	46
5.3	Regeneration of XFS UUID	46
5.4	Direct Selection of Backup StorageClass	47
5.5	Direct Node Selection when Attaching the Longhorn Cloned PVC	47
5.6	Using a StorageClass with Binding Mode “WaitForFirstConsumer”	47
5.7	Retry Standard Backup if Clone Backup Fails	47
6	OpenShift: Limitations	47
7	OpenShift: Troubleshooting	48
7.1	Error: kubernetes: incluster error: Service host/port is not set	48

Contents

Note

You can download this article as a [PDF](#)

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

Containers are very light system level virtualization with less overhead because programs in virtual partitions use the operating system's normal system call interface and do not need to be subjected to emulation or run in an intermediate virtual machine. OpenShift manages a set of containers to create a flexible execution environment for applications and services.

The Bacula Enterprise OpenShift Plugin will save all the important OpenShift resources which make up the application or service. This includes the following name-spaced objects:

- Config Map
- Daemon Set
- Deployment
- Endpoint
- Limit Range
- Pod
- Persistent Volume Claim
- Pod Template
- Replica Set
- Replication Controller
- Resource Quota
- Route
- Secret
- Service
- Service Account
- Stateful Set
- PVC Data Archive

Note

The PVC Data is not exact OpenShift Object but represents archive of real data existed on selected PVC.

and non namespaced objects:

- Namespace
- Persistent Volume
- Storage Class

All name-spaced objects which belong to the particular namespace are grouped together for easy backup data browsing and recovery.

Users and service accounts can be authorized to access the server API. This process goes through authentication, authorization and admission control. To be able to successfully backup the OpenShift resources, it is required to have a user or service account with the correct permissions and rights to be successfully authenticated and authorized to access the API server and resources to be backed up.

For resource configuration backups, the user or service account must be able to read and list the resources. In the case of PVC Data backup, it is also required that the user or service account can create and delete pods because the plugin will need to create and delete the Bacula Backup Proxy Pod during the backup.

Please see the OpenShift documentation for more details.

1 OpenShift: Features

[CommunityEnterprise](#) The article aims at presenting the OpenShift Plugin features.

Note

Only Full level backups are supported.

They are:

- OpenShift cluster objects configuration backup
- OpenShift Virtualization cluster objects configuration backup
- Ability to restore single configuration object in OpenShift/OpenShift Virtualization
- Ability to restore object configuration to local directory in OpenShift/OpenShift Virtualization
- Ability to restore OpenShift/OpenShift Virtualization object configurations to a different OpenShift namespace, name or StorageClass
- Persistent Volumes data backup and restore for OpenShift
- OpenShift Etcd backup, also compatible with HA cluster
- Ability to restore Persistent Volumes data to local directory in OpenShift
- Ability to restore OpenShift Persistent Volumes data to a different OpenShift namespace, name or StorageClass
- Ability to use CSI driver volume snapshot and cloning features in OpenShift to perform Persistent Volume data backup
- Ability to use Longhorn snapshots to perform Persistent Volume data backup
- Ability to adapt the best technique to each Persistent Volumes data backup
- Ability to execute user defined commands on required Pod containers to prepare and clean data backup
- Configure OpenShift workload backup requirements directly with Pod annotations.
- Automatic cleanup of pods and cloned PVCs from previous backup jobs that did not complete successfully
- Support for inline compression of block-mode Persistent Volume Claims during backup
- Ability to back up a single OpenShift Virtualization virtual machine along with its dependencies

- Ability to list all OpenShift/OpenShift Virtualization objects
- Ability to automatically start a restored virtual machine after the restore process completes

2 OpenShift: Installation

CommunityEnterprise The Bacula File Daemon and the OpenShift Plugin can be installed outside of the OpenShift cluster on a server which has access to the OpenShift API, or inside a protected cluster in a Container/Pod. The OpenShift Plugin can be installed on different operating systems and distributions, so the Bacula Enterprise File Daemon for the correct operating system and platform has to be used.

There is no need, or in some solutions (when K8S is a cloud service like GKE or EKS), it is even a possible to install a Bacula File Daemon and the OpenShift Plugin on a OpenShift Master Server (etcd, control pane).

2.1 Prerequisites

The **Plugin Directory** directive of **File Daemon** resource in `/opt/bacula/etc/bacula-fd.conf` must point to where the `kubernetes-fd.so` plugin file is installed. The standard Bacula plugin directory is `/opt/bacula/plugins`

```
FileDaemon {
  Name = bacula-fd
  Plugin Directory = /opt/bacula/plugins
  ...
}
```

2.2 Installation with Package Manager

CommunityEnterprise Installation of the Bacula Enterprise OpenShift Plugin is most easily done by adding the OpenShift repository from your Bacula Enterprise subscription into the package manager configuration of your Linux distribution.

An example would be `/etc/apt/sources.list.d/bacula.list` for Debian-based Linux distributions. The contents of this file appear as follows:

```
# Bacula Enterprise
deb https://www.baculasystems.com/dl/@cust@/debs/bin/@ver@/stretch-64/
↳bookworm main
deb https://www.baculasystems.com/dl/@cust@/debs/kubernetes/@ver@/stretch-64/
↳bookworm kubernetes
```

where “@cust@” is your individual download area identification string found in your Welcome Package.

Subsequently, it is necessary to execute the `apt-get update` command. Then, the OpenShift Plugin can be installed using: `apt-get install bacula-enterprise-kubernetes-plugin`.

On RHEL, extend the repository file for your package manager to contain a section for the plugin - `/etc/yum.repos.d/bacula.repo`:

```
[Bacula]
name=Bacula Enterprise
baseurl=https://www.baculasystems.com/dl/@cust@/rpms/bin/@version@/rhel8-64/
enabled=1
protect=0
```

(continues on next page)

(continued from previous page)

```
gpgcheck=1
[Bacula EnterpriseDockerPlugin]
name=Bacula Enterprise Kubernetes Plugin
baseurl=https://www.baculasystems.com/dl/@cust@/rpms/kubernetes/@version@/
↳rhel8-64/
enabled=1
protect=0
gpgcheck=1
```

Next, perform a `yum update` to prepare your system, after which you can install the OpenShift Plugin package by running `yum install bacula-enterprise-kubernetes-plugin`.

Manual installation of the packages can be performed after downloading the required files from your Bacula Systems download area, and then using the low-level package manager tools (`rpm` or `dpkg`) to perform the plugin installation.

2.3 bacula-backup Proxy Pod Image Installation

CommunityEnterprise For OpenShift PVC Data backup functionality, the Bacula Enterprise OpenShift Plugin requires a dedicated Bacula Backup Proxy Pod. This Pod is deployed using an image that is available in the `bacula-enterprise-kubernetes-tools` package.

You must install this image manually into the local Docker image registry service used by your OpenShift cluster as a source for application images.

Installation of the image can be performed with the following example commands:

```
# cd /opt/bacula/lib
# docker load -i bacula-backup-<timestamp>.tar
# docker image tag bacula-backup:<timestamp> <registry>/bacula-backup:
↳<timestamp>
# docker push <registry>/bacula-backup:<timestamp>
```

where `<timestamp>` is the image version shipped with above package and `<registry>` is the location of your Docker image registry service. The exact procedure depends on your OpenShift cluster deployment, so verify the steps above before running the Docker commands.

You can use any registry service available to your cluster, public or private, e.g. `gcr.io/`.

Depending on your cluster configuration, it may be necessary to set the `baculaimage=<name>` plugin parameter (see this section for details) to define which repository and Container image to use. The default for this parameter is `bacula-backup:<timestamp>` which may not be correct for your deployment.

Another example where you must modify the Bacula Backup Proxy Pod Image is when your registry requires authentication. Click here for more details.

2.4 Advanced Bacula Backup Proxy Pod Deployment

CommunityEnterprise

Warning

This is an advanced topic related to OpenShift clusters. It is strongly advised against attempting to modify or configure the Bacula OpenShift Plugin unless you have a deep understanding of the process.

You can customize the service parameters used to deploy Bacula backup Pods dedicated to Persistent Volume Claim data backup. The plugin uses the following Pod service deployment YAML template to execute the proxy operation pod in the cluster.

```
apiVersion: v1
kind: Pod
metadata:
  name: {pod_name}
  namespace: {namespace}
  labels:
    app: baculabackup
spec:
  hostname: {pod_name}
  {node_name}
  containers:
  - name: {pod_name}
    {security_context}
    resources:
      limits:
        cpu: "1"
        memory: "64Mi"
      requests:
        cpu: "100m"
        memory: "16Mi"
    image: {image}
    env:
      - name: PLUGIN_DEBUG
        value: "{plugin_debug}"
      - name: PLUGIN_MODE
        value: "{mode}"
      - name: PLUGIN_COMPRESSION_IN_CONTAINER
        value: "{compression_in_container}"
      - name: PLUGIN_HOST
        value: "{host}"
      - name: PLUGIN_PORT
        value: "{port}"
      - name: PLUGIN_TOKEN
        value: "{token}"
      - name: PLUGIN_JOB
        value: "{job}"
    imagePullPolicy: {image_pull_policy}
    {volume_attachments}
    {image_pull_secrets}
    restartPolicy: Never
    {volumes}
```

The template above contains predefined placeholders that are replaced with the corresponding values during Pod execution preparation. To customize proxy Pod deployment, you can tune the template variables or modify the template body. The list below describes all supported variables and their requirements.

pod_name

This is the predefined Pod name used by the plugin. This variable is required and cannot be customized.

namespace

This is a Namespace name for which the PVC Data backup is performed. This variable is required and cannot be customized.

node_name

This is a placeholder for Cluster node name parameter (`nodeName: ...`) used to mount an existing Volume Claim for backup or restore, if required for the selected PVC. In most cases, this variable is required and cannot be customized.

security_context

In some environments, a specific security context is required to run as the root user. This context corresponds to the `security_context=...` OpenShift Plugin parameter. This variable is optional.

image

This is a Pod Container image name to execute. You can customize or omit this variable as long as you provide a Container image name required by the cluster.

plugin_debug

This is the debug level that should be set in the container. This corresponds to the `debug=...` OpenShift Plugin parameter. This variable is optional.

mode

This is an operation mode for the Proxy Pod. Supported values are: `backup` and `restore`. This variable is required and cannot be customized.

compression_in_container

This is the algorithm to compress data used when the pvc is in block mode. This corresponds to the `compression_in_container=...` OpenShift Plugin parameter. This variable is optional.

host

This is an endpoint address which corresponds to the `pluginhost=...` OpenShift Plugin parameter. This variable is required. You can customize or omit this variable as long as you provide a value for the **PLUGIN_HOST** Container environment.

port

This is an endpoint address which corresponds to the `pluginport=...` OpenShift Plugin parameter. This variable is required. You can customize or omit this variable as long as you provide a value for the **PLUGIN_PORT** Container environment.

token

This is an authorization token (randomly generated). This variable is required and cannot be customized.

job

This is the job name. This variable is required and cannot be customized.

image_pull_policy

This is the policy to pull image from authenticated registry. This corresponds to the `imagepullpolicy=...` OpenShift Plugin parameter. This variable is optional.

volume_attachments

This is the pvc attachment to the Pod. This variable is required and cannot be customized.

image_pull_secrets

This is the name of the OpenShift Secret used to pull image from authenticated registry. This corresponds to the `imagepullsecrets=...` OpenShift Plugin parameter. This variable is optional.

volumes

This is the name of the PVC for backup or restore operations. This variable is required and cannot be customized.

You can create the required file: `/opt/bacula/scripts/bacula-backup.yaml` or point to a custom one using the `$DEFAULTPODYAML` environment variable.

2.5 Longhorn Snapshot Installation

CommunityEnterprise Starting with Bacula Enterprise **18.1.0**, support for Longhorn snapshot functionality was introduced, along with various other features. The plugin automatically detects its ability to perform this functionality.

Longhorn is a lightweight, distributed block storage system designed for Kubernetes/OpenShift. Developed by Rancher Labs, it provides a highly available solution for managing persistent volumes. Longhorn enables the creation, management, and recovery of data volumes with ease, ensuring data integrity and availability even in the event of node failures. [More information about Longhorn.](#)

This storage type supports Kubernetes/OpenShift CSI snapshot mechanism, see [CSI snapshot support](#). If you do not want to integrate with this technology, you can still perform snapshot-based backups using Longhorn through this plugin. To do so, follow the steps below.

Longhorn Snapshot Backup Requirements

For Longhorn Snapshot Backup functionality, the Bacula Enterprise Kubernetes/OpenShift Plugin uses [Longhorn Python Client](#). It requires access to Longhorn endpoint. One way to communicate with Longhorn is through the `longhorn-frontend` service.

If you run Bacula Enterprise Kubernetes/OpenShift Plugin inside the cluster, connect to: `http://longhorn-frontend.longhorn-system/v1`.

If you run Bacula Enterprise Kubernetes/OpenShift Plugin outside the cluster and have `kubectl` installed, you can use `kubectl port-forward` to forward the `longhorn-frontend` service to localhost:

```
kubectl port-forward services/longhorn-frontend 8080:http -n longhorn-system
```

Then connect to endpoint: `http://localhost:8080/v1`.

If you have not installed `kubectl`, you can create an Ingress resources in the `longhorn-system` namespace to access the `longhorn-frontend` service:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: longhorn-ingress
  namespace: longhorn-system
annotations:
  nginx.ingress.kubernetes.io/ssl-redirect: "false"
  # If you want to limit the access to Bacula-FD IP (Example: "192.168.1.34")
  nginx.ingress.kubernetes.io/whitelist-source-range: "192.168.1.34/32"
spec:
  rules:
    - host: longhorn-frontend.testing-cluster.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: longhorn-frontend
```

(continues on next page)

(continued from previous page)

```
port:  
  number: 80
```

Then connect to: `http://longhorn-frontend.testing-cluster.com/v1`.

You must also configure a Backup Target in Longhorn. Follow the steps in: [Setting a Backup Target in Longhorn](#).

3 OpenShift: Configuration

CommunityEnterprise The plugin is configured using **Plugin Parameters** defined in a **Filesets** -> **Include** section of the Bacula Enterprise Director configuration. Starting from Bacula Enterprise version **16.0.7**, it is now possible to enclose particular parameters of this plugin within single quotation marks for delimitation purposes.

3.1 Generic Plugin Parameters

CommunityEnterprise The following OpenShift Plugin parameters affect any type of Job (Backup, Estimate, or Restore).

abort_on_error[=**<0 or 1>**]

specifies whether or not the plugin should abort execution (and the Bacula Job) if a fatal error occurs during a Backup, Estimate, or Restore operation. If not specified, the default value is 0.

This parameter is optional.

config=**</path/to/file>**

specifies the path to the configuration file used to connect to the OpenShift cluster. Refer to the OpenShift documentation for more information about the *kubeconfig* file. If this directive is omitted and no other access method is configured, the default configuration file location will be used: `$HOME/.kube/config`.

This parameter is optional.

incluster

if defined, standard in-cluster access to the OpenShift API is used. This option should be used only when the Bacula File Daemon and OpenShift Plugin are deployed as a OpenShift cluster service in a Pod. In this case, OpenShift automatically provides the required access credentials. This option does not work when executed outside of a OpenShift cluster.

This parameter is optional.

host=**<url-k8s-api>**

defines a OpenShift API url. This option must be used together with the token parameter described below. If this option is specified, then both parameters are required.

This parameter is optional.

token=**<bearer-token>**

defines a Bearer token used for authorization to the OpenShift API defined by **host**=**<url-k8s-api>**. This option must be used together with the host parameter described above. You can read more about Bearer token authentication here: <https://swagger.io/docs/specification/authentication/bearer-authentication/>

This parameter is optional.

verify_ssl[=**<0 or 1>**]

specifies whether the plugin should verify a OpenShift API certificate when the connection uses

SSL/TLS. If set to **verify_ssl=0**, verification is disabled. This can be useful when connecting to a OpenShift API server that uses a self-signed certificate. By default, if this parameter is not set, proper certificate validation is performed by default.

This parameter is optional.

ssl_ca_cert=</path/to/file>

specifies a file with the CA certificate used to customize the OpenShift API server identity to verify the peer.

This parameter is optional.

debug=[1,2,3]

enables generation of a backend debug file at the following location: `/bacula/working/kubernetes/`. The debug file can assist with troubleshooting job execution issues. If not defined, no debug file is generated. Starting from version 16.0.14, multiple debug levels are available. Each higher level includes the information from the previous levels.

1: Save debug messages of k8s server interactions. **2:** Save communication messages with bacula core except data packages. From version **18.0.9**, this debug level also includes relevant information about the OpenShift environment like: namespaces, storage classes, pvcs and their pvs. **3:** Save all debug messages.

This parameter is optional. Normally, the level 2 is enough to open support cases.

3.2 Estimate and Backup Plugin Parameters

CommunityEnterprise These plugin parameters are relevant only for Backup and Estimate jobs:

Cluster Resources

namespace=<name>

specifies a OpenShift namespace name to be backed up. Multiple **namespace=<name>** parameters are allowed if you want to backup multiple namespaces. If a namespace with the specified **name** cannot be found, it will be silently ignored. If this parameter is not set, all the objects in all namespaces will be saved. When performing a pvdata backup, the namespace(s) must be specified. The Persistent Volume(s) in the specified namespace(s) will be included in the backup.

This parameter is required for pvdata backup. Otherwise, this parameter is optional.

persistentvolume=<name>

specifies a OpenShift Persistent Volume configuration to be backed up. Multiple **persistentvolume=<name>** parameters are allowed if you want to back up multiple volumes. Standard shell wildcard pattern matching can be used to select multiple volumes with a single **persistentvolume** parameter. If a persistent volume with the specified **name** cannot be found, it will be silently ignored. If this parameter is not set, all Persistent Volume configurations will be saved unless **pvconfig=0** is set as described below or **pvdata** is defined. If the **pvdata** parameter is defined (in any form), only the Persistent Volumes linked to the defined pvdata will be backed up.

This parameter is optional.

pvconfig=[0|1]

disables Persistent Volume configuration backups when **pvconfig=0** is set. By default, Persistent Volume configurations are backed up.

This parameter is optional.

Note

Starting with Bacula Enterprise 18.1, if the parameter `pvdata` is set, the plugin exclusively performs backups of the required Persistent Volume configuration.

storageclass=<name>

specifies a OpenShift Storage Class configuration to be backed up. Multiple `storageclass=<name>` parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple volumes using a single `storageclass` parameter. If a storage class with the specified `name` cannot be found, it will be silently ignored. If this parameter is not set, all storage class configuration will be saved unless `scconfig=0` is set as described below.

This parameter is optional.

scconfig=[0|1]

disables Storage Class configuration backups when `scconfig=0` is set. By default, StorageClass object configurations are backed up.

This parameter is optional.

clusterrole=<name>

specifies a OpenShift Cluster Role configuration to be backed up. Multiple `clusterrole=<name>` parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple cluster roles using a single `clusterrole` parameter. If a cluster role with `name` cannot be found, it will be silently ignored. If this parameter is not set, all Cluster Role and Cluster Role Binding configuration will be saved unless `crconfig=0` is set as described below.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

crconfig=[0|1]

disables Cluster Role and Cluster Role Binding configuration backups when `crconfig=0` is set. By default, ClusterRole object configurations are backed up.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

clusterrolebinding=<name>

specifies a OpenShift Cluster Role Binding configuration to be backed up. Multiple `clusterrolebinding=<name>` parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple cluster role bindings using a single `clusterrolebinding` parameter. If a cluster role binding with `name` cannot be found, it will be silently ignored. If this parameter is not set, all Cluster Role Binding configuration will be saved unless `crbconfig=0` is set as described below.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

crbconfig=[0|1]

disables Cluster Role Binding configuration backups when **crbconfig=0** is set. The default is to backup Cluster Role Binding object configurations.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

migration_policy=<name>

specifies a KubeVirt Migration Policy configuration to be backed up. Multiple **migration_policy=<name>** parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple migration policies using a single **migration_policy** parameter. If a Migration Policy with **name** can not be found, it will be silently ignored. If this parameter is not set, all Migration Policy configuration will be saved unless **vmcf_config=0** is set as described below.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

mp_config=[0|1]

disables Migration Policy configuration backups when **mp_config=0** is set. By default, MigrationPolicy object configurations are backed up.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

virtual_machine_cluster_instance_type=<name>

specifies a KubeVirt Virtual Machine Cluster Instance Type configuration to be backed up. Multiple **virtual_machine_cluster_instance_type=<name>** parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple virtual machine cluster instance types using a single **virtual_machine_cluster_instance_type** parameter. If a virtual machine cluster instance type with **name** cannot be found, it will be silently ignored. If this parameter is not set, all Virtual Machine Cluster Instance Type configuration will be saved unless **vmcf_config=0** is set as described below.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

vmcf_config=[0|1]

disables Virtual Machine Cluster Instance Type configuration backups when **vmcf_config=0** is set. By default, Virtual Machine Cluster Instance Type object configurations are backed up.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

virtual_machine_cluster_preference=<name>

specifies a KubeVirt Virtual Machine Cluster Preference configuration to be backed up. Multiple **virtual_machine_cluster_preference=<name>** parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple Virtual Machine Cluster Preferences using a single **virtual_machine_cluster_preference** parameter. If a virtual machine cluster Preference with **name** cannot be found, it will be silently ignored. If this parameter is not set, all Virtual Machine Cluster Preference configuration will be saved unless **vmcp_config=0** is set as described below.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.1.

vmcp_config=[0|1]

disables Virtual Machine Cluster Preference configuration backups when **vmcp_config=0** is set. By default, Virtual Machine Cluster Preference object configurations are backed up.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.2.

volume_snapshot_class=<name>

specifies a OpenShift Volume Snapshot Class configuration to be backed up. Multiple **volume_snapshot_class=<name>** parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple Volume Snapshot Classes using a single **volume_snapshot_class** parameter. If a Volume Snapshot Class with **name** cannot be found, it will be silently ignored. If this parameter is not set, all Volume Snapshot Class configuration will be saved unless **vsc_config=0** is set as described below.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.2.

vsclass_config=[0|1]

disables Volume Snapshot Content configuration backups when **vsclass_config=0** is set. By default, VolumeSnapshotClass object configurations are backed up.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.2.

volume_snapshot_content=<name>

specifies a OpenShift Volume Snapshot Content configuration to be backed up. Multiple **volume_snapshot_content=<name>** parameters are allowed if you want to backup multiple objects. Standard shell wildcard pattern matching can be used to select multiple Volume Snapshot Contents using a single **volume_snapshot_content** parameter. If a Volume Snapshot Content with name cannot be found, it will be silently ignored. If this parameter is not set, all Volume Snapshot Content configuration will be saved unless **vsc_config=0** is set as described below.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.2.

vsc_config=[0|1]

disables Volume Snapshot Content configuration backups when **vsc_config=0** is set. By default, VolumeSnapshotContent object configurations are backed up.

This parameter is optional.

Note

This parameter is available since Bacula Enterprise 18.2.

Namespaced Resources

backup_mode=[snapshot|clone|standard]

sets the mode to perform backup of the persistent volume data (pvdata backup). When this parameter is specified, the selected mode must be compatible with the underlying storage. If the Persistent Volume is not compatible with the backup mode specified (snapshot or clone), the plugin will try another backup mode in this order: Clone and Standard.

Note

The **bacula/backup.mode** annotation in the OpenShift resource takes precedence over this parameter for the PVCs defined in the pod.

Note

The plugin automatically verifies whether the selected backup mode is compatible before starting the job. However, this does not mean that it will automatically switch to another mode if a failure occurs. If a backup fails while running in snapshot, clone, or standard mode, the plugin will not retry the operation using a different mode.

pvcdata[=<pvcname1,pvcname2,...>]

specifies OpenShift Persistent Volume Claim names for which a PVC Data archive will be created. Since PVCs are namespaced objects, you must specify the required namespace(s) with the **namespace=<name>** parameter. If the **pvcdata** parameter is defined without an equals sign (“=”) and subsequent value, all detected PVCs in the specified namespace(s) will be selected for the PVC Data archive backup. You can select several Persistent Volume Claim in this parameter.

This parameter is optional.

Filtering Resources

For example, if we apply the following filters:

filter_pvcs_by_storage_class=longhorn and **filter_pvcs_by_volume_mode=block**, only the PVCs that meet both conditions will be backed up.

However, if we apply: **filter_pvcs_by_storage_class=longhorn,ceph**, then PVCs that belong to storage-Class **longhorn** as well as those that belong to **ceph** will be backed up.

filter_pvcs_by_storage_class[=<storage_class_1,storage_class_2,...>]

when set, only PVCs whose StorageClass name matches the provided filter are included in the backup; all other PVCs are excluded.

This parameter is optional.

filter_pvcs_by_volume_mode[=block|filesystem]

when set, only PVCs whose Volume mode matches the provided filter are included in the backup; all other PVCs are excluded.

This parameter is optional.

Note

- Different filter types behave as a logical AND
- Multiple values within the same filter type behave as a logical OR.

exclude_namespaced_resources[=pods|secrets|configmaps...]

excludes namespaced resources by the resource category. The resource category is the name used by OpenShift.

Note

Persistent Volume Claims cannot be excluded.

Etcd Backup

etcd_backup=[snapshot]

enables **etcd** to be backed up. If this parameter is omitted, **etcd** is not backed up.

This parameter is optional.

etcd_namespace=<name>

specifies the OpenShift namespace name assigned to **etcd**. If this parameter is not defined, the default **kube-system** is used.

This parameter is optional and applies when **etcd_backup** is enabled.

etcd_api=<number>

specifies the API version that the `etcdctl` binary is required to operate with. If this parameter is not defined, the default **3** will be used.

This parameter is optional and applies when **etcd_backup** is enabled.

etcd_endpoints=<http://static-ip:etcd-port,https://domain-name:etcd-port>

defines the **etcd** endpoints that are utilized for operation. If the OpenShift cluster has multiple **etcd** nodes, you must concatenate all **etcd** nodes.

This parameter depends on the activation of **etcd_backup**.

etcd_secret=<name>

defines the OpenShift Secret which stores the **etcd** certificates and key. This OpenShift Secret must be created in the defined **etcd_namespace** (by default `kube-system`).

This parameter depends on the activation of **etcd_backup**. This parameter is optional if you use the parameters **etcd_ca_cert**, **etcd_tls_cert** and **etcd_tls_key**.

etcd_ca_cert=</path/to/file>

is the CA certificate file location used to access to **etcd**.

This parameter depends on the activation of **etcd_backup**. This parameter is optional if you use the parameter **etcd_secret**.

etcd_tls_cert=</path/to/file>

is the TLS certificate file location used to access to **etcd**.

This parameter depends on the activation of **etcd_backup**. This parameter is optional if you use the parameter **etcd_secret**.

etcd_tls_key=</path/to/file>

is the TLS key file location used to access to **etcd**.

This parameter depends on the activation of **etcd_backup**. This parameter is optional if you use the parameter **etcd_secret**.

If none of the parameters above are specified, then all available Namespaces and Persistent Volume Configurations will be backed up. However, the plugin does not force a PVC Data archive backup neither **etcd** backup in this case.

longhorn_system=<longhorn-namespace>

defines the namespace where Longhorn is deployed. Longhorn documentation recommends using **longhorn_system**. The default value of this parameter is **longhorn_system**.

This parameter is optional and required only when performing a CSI snapshot backup with Longhorn as the provider.

longhorn_url=<longhorn-endpoint>

defines the Longhorn UI endpoint to manage Longhorn storages. For more information, refer to the Longhorn Snapshot Installation section.

This parameter is optional and required only when performing a snapshot backup with Longhorn without the CSI driver.

3.3 Backup and Restore Plugin Parameters

CommunityEnterprise These plugin parameters are relevant only for Backup and Restore jobs when the PVC Data archive functionality is used:

fdaddress=<IP or name>

is the IP address or host name where the OpenShift Plugin should listen for incoming connections from a Bacula Backup Proxy Pod. This parameter, combined with **fdport=<number>** below will define a socket to listen on. The default is to listen on all available interfaces (0.0.0.0) which should be fine in most cases.

This parameter is optional if **pluginhost=<IP or name>** is defined.

fdport=<number>

is a port number on which OpenShift Plugin should listen for incoming connections from a Bacula Backup Proxy Pod. This parameter, combined with **fdaddress=<IP or name>** above will define a socket to listen on. The default is to listen on port 9104.

This parameter is optional.

pluginhost=<IP or name>

is the entry point address where a Bacula Backup Proxy Pod should connect during backup or restore operations. The name should be resolvable on the OpenShift cluster, otherwise an IP address must be used here. This with **pluginport=<number>** parameter below will define the exact service entry point. The default is to use the value from **fdaddress=<IP or name>** parameter above.

This parameter is required when **fdaddress=<IP or name>** is not defined. Otherwise it is optional.

pluginport=<number>

is the port number for service entry point address where the Bacula Backup Proxy Pod should connect during backup or restore operations. This, combined with the **pluginhost=<IP or name>** parameter above define the exact service entry point. The default is to use the value from the **fdaddress=<IP or name>** parameter above. When neither is defined the default 9104 port number will be used.

This parameter is optional.

fdcertfile=<path>

is the SSL certificate file location for the OpenShift Plugin endpoint service data connection for a Bacula Backup Proxy Pod. The certificate and key (see **fdkeyfile=<path>** below) files are required for proper Bacula Backup Proxy Pod. You can create and use any valid SSL certificate including a self-signed one. You can even just use the certificate generated during the OpenShift Plugin installation located at `/opt/bacula/etc/bacula-backup.cert`. The default is to use the certificate generated during plugin installation.

This parameter is optional.

fdkeyfile=<path>

is an SSL private key file location for the SSL certificate defined by **fdcertfile=<path>** above.

An unencrypted private key must be used for this purpose. The default is to use a private key file created during plugin installation at `/opt/bacula/etc/bacula-backup.key` when **fdcertfile=<path>** above is not defined (the default). Otherwise the plugin will refer to the same certificate file from **fdcertfile=<path>** and use it as a `.pem` Container.

This parameter is optional.

baculaimage=<name>

is a Bacula Backup Proxy Pod Container image registry location for your cluster as described in the image installation chapter **baculatarimage**. In most cases this parameter will consist of your Docker images registry location with the tag of the required image. The default for this parameter is `bacula-backup:<timestamp>` and may not match your cluster configuration.

This parameter is optional.

imagepullpolicy=<Always|Never|ifNotPresent>

sets the OpenShift pod image pull policy during Bacula Backup Proxy Pod Container deployment. You can configure OpenShift to **Always** or **Never** pull the image from the repository, or pull it only when not available with **ifNotPresent**. The option value is case insensitive. If this parameter is not defined, the default **ifNotPresent** will be used.

This parameter is optional.

imagepullsecrets=<secret>

sets the OpenShift image pull secrets during Bacula Backup Proxy Pod Container deployment. You need to create the secret in all namespaces where you want to perform a backup. If this parameter is not defined, you can not pull any image from an authenticated registry.

This parameter is optional. This parameter is available for use starting from version **18.1.0**. Before version 18.0.1, it is necessary to set the Container deployment. To learn more about this method, see: [Advanced Bacula Backup Proxy Pod Deployment page](#).

compression_in_container=[raw|gzip|xz|zstd|lz4]

sets the inline compression during backup and restore of block Persistent Volume Claims. If this parameter is not defined, the default **raw** will be used.

This parameter is optional.

security_context=[0|1]

sets the OpenShift parameter `securityContext` when running the Bacula Backup Pod. It may be required in certain environments.

3.4 Restore Plugin Parameters

CommunityEnterprise During restore, the OpenShift Plugin will use the same parameters which were set for the backup job and saved in the catalog. During restore, you may change any of the parameters described in chapter `generic8spluginparameters` and `k8srbackupparameters`. In addition to the options used for backups, the **outputformat** option can be used during restore. This option specifies the file format when restoring to a local filesystem. You can choose the restore output in JSON or YAML. If not defined the restored files will be saved in YAML.

outputformat=[json|yaml]

specifies the file format when restoring to a local filesystem as described above.

This parameter is optional.

new_namespace=<namespace>

specifies an alternative namespace to restore the selected objects. If the namespace doesn't exist, the OpenShift Plugin will create a new one.

new_name=<name>

specifies an alternative name to restore the selected objects. We advise using it solely when you choose a single resource.

new_storageclass=<storageclass_name>

specifies an alternative `storageclass` name to restore the selected Persistent Volume Claim along with its associated data.

mac_address=[keep|new|MAC]

specifies whether to retain the existing MAC address or assign a new MAC address to all VM network interfaces.

run_after_restore=[yes|no]

specifies whether the virtual machine should be started automatically after the restore process completes.

3.5 Tuning Plugin Parameters

CommunityEnterprise These parameters are available starting from Bacula version 18.2.0.

They apply to all operations and are considered advanced settings. In general, they should not be modified and are therefore optional.

They can be used to fine-tune the plugin's behavior in specific scenarios, such as unstable network environments or other special deployment conditions.

Warning Suppression

The following parameters are relevant to suppress plugin warning.

suppress_warnings_all=[0|1]

specifies whether warning messages should affect the final status of the Bacula Job. The default value is 1.

suppress_warnings_clone_file_check=[0|1]

specifies whether warnings generated during clone backup verification should affect the final status of the Bacula Job. The default value is 1.

suppress_warnings_fs_matching_pv_pvc=[0|1]

specifies whether warnings generated when checking filesystem compatibility between PVCs and PVs should affect the final job status. The default value is 1.

Timeouts

The following parameters are relevant to determine a timeout for specific actions.

timeout=<seconds>

specifies the number of seconds allowed for various network operations. Examples include: waiting for Bacula Backup Proxy Pod connection or OpenShift API operations, waiting for Pod execution or removal. The default is 600 seconds. The minimum allowed value is 100 seconds. If an invalid value is specified, the default value is used.

timeout_bacula_pod_deploy=<seconds>

specifies the maximum number of seconds allowed to deploy the bacula-backup pod. The default is 600 seconds.

timeout_bacula_pod_delete=<seconds>

specifies the maximum number of seconds allowed to delete the bacula-backup pod. The default is 600 seconds.

timeout_bacula_pod_communication=<seconds>

specifies the maximum number of seconds allowed without receiving data from the bacula-backup pod. The default is 600 seconds.

Intervals

The following parameters define intervals for specific operations.

interval_bacula_pod_is_ready=<seconds>

specifies the number of seconds to check if the bacula-backup pod is ready. The default is 1 second.

interval_bacula_pod_is_deleted=<seconds>

specifies the number of seconds to check if the bacula-backup pod has been deleted. The default is 1 second.

interval_bacula_pod_get_description=<seconds>

specifies the number of seconds to retrieve status information about the bacula-backup pod. The default is 100 seconds.

interval_bacula_pod_redeploy=<seconds>

specifies the number of seconds to wait before removing and redeploying the backup-backup pod. The default is 300 seconds.

Max Reruns

The following parameter defines the maximum number of retry attempts before reporting an error.

max_rerun_bacula_pod_deploy=<number>

specifies the maximum number of attempts to redeploy the bacula-backup pod. The default is 5 tries.

3.6 Fileset Examples

CommunityEnterprise In the example below, all OpenShift Namespaces, Objects and Persistent Volume Configurations will be backed up using the default OpenShift API access credentials.

```
FileSet {
  Name = FS_Kubernetes_All
  Include {
    Plugin = "kubernetes:"
  }
}
```

In this example, we will backup a single OpenShift Namespace using the Bearer Token authorization method.

```
FileSet {
  Name = FS_Kubernetes_plugintest
  Include {
    Plugin = "kubernetes: host=http://10.0.0.1/k8s/clusters/test \
      token=kubeconfig-user:cbhssdxq8vv8hrcw8jdxs2 namespace=plugintest"
  }
}
```

The same example as above, but with a Persistent Volume:

```
FileSet {
  Name = FS_Kubernetes_mcache1
  Include {
    Plugin = "kubernetes: host=http://10.0.0.1/k8s/clusters/test \
            token=kubeconfig-user:cbhssdxq8vv8hrcw8jdxs2 \
            namespace=plugintest persistentvolume=myvol"
  }
}
```

This example backs up a single Namespace and all detected PVCs in this Namespace using a defined listening and entry point address and the default connection port:

```
FileSet {
  Name = FS_Kubernetes_test_namespace
  Include {
    Plugin = "kubernetes: namespace=test pvcdta fdaddress=10.0.10.10"
  }
}
```

The same example as above, but using different listening and entry point addresses as may be found when the service is behind a firewall using port forwarding features:

```
FileSet {
  Name = FS_Kubernetes_test_namespace_through_firewall
  Include {
    Plugin = "kubernetes: namespace=test pvcdta=plugin-storage fdaddress=10.
↪0.10.10 \
            pluginhost=backup.example.com pluginport=8080"
  }
}
```

This example shows PVC Data archive backup with the Bacula File Daemon inside a OpenShift cluster:

```
FileSet {
  Name = FS_Kubernetes_incluster
  Include {
    Plugin = "kubernetes: incluster namespace=test pvcdta \
            pluginhost=backup.bacula.svc.cluster.local"
  }
}
```

The configuration above is designed for use in situations where the Bacula server components are located on-premise and behind a firewall with no external ports allowed in, but must back up data on an external OpenShift cluster.

In the following example, we present a set of filesets designed to implement parallel backups in the same namespace. We will backup the persistent volume claims, specifically pvc1 and pvc2, which both belong to same namespace (ns-1):

```
Fileset {
  Name = "Test-K8S-Set-0006-5"
  Include { Options { signature=SHA1 }
    Plugin = "kubernetes: pluginhost=10.255.3.208 pvcdta=pvc1 namespace=ns-1
↪pluginport=9104 fdport=9104"
```

(continues on next page)

(continued from previous page)

```
}  
}  
  
Fileset {  
  Name = "Test-K8S-Set-0006-6"  
  Include { Options { signature=SHA1 }  
    Plugin = "kubernetes: pluginhost=10.255.3.208 pvcdata=pvc2 namespace=ns-1  
↳pluginport=9105 fdport=9105"  
  }  
}
```

Note

It is necessary to verify that the `fdport` is not being used by any other process.

4 OpenShift: Operations

[CommunityEnterprise](#) The following article aims at describing Kubernetes OpenShift operations.

4.1 Backup

[CommunityEnterprise](#) The plugin can back up a number of OpenShift objects including: Deployments, Pods, Services or Persistent Volume Claims, check the following chapter for a complete list.

The backup will create a single (`.yaml`) file for any OpenShift object which is saved. The OpenShift Plugin provides a data backup feature for Filesystem PVCs by creating a data archive in the form of a single file named `<pvc_name>.tar` or `<pvc_name>.<compression_type>` for block data. For the Etcd backup capability, it generates a snapshot with the format `snapshot-<day>-<month>-<year>.db`. The objects are organized within the Bacula Catalog to facilitate browsing and restore operations. In the Bacula Catalog, OpenShift objects are represented as follows:

- `/@kubernetes/namespaces/<namespace>.yaml` Namespace definition
- `/@kubernetes/namespaces/<namespace>/<objecttype>/<name>.yaml` Object definitions in the namespace
- `/@kubernetes/namespaces/<etcd-namespace(kube-system?)>/etcd/snapshot-<day>-<month>-<year>.db` Etcd snapshot in their namespace
- `/@kubernetes/namespaces/<namespace>/persistentvolumeclaims/<pvc_name>.tar` Filesystem PVC Data backup in the selected namespace
- `/@kubernetes/namespaces/<namespace>/persistentvolumeclaims/<pvc_name>.raw` Block PVC Data backup in the selected namespace without inline compression
- `/@kubernetes/namespaces/<namespace>/persistentvolumeclaims/<pvc_name>.gz` Block PVC Data backup in the selected namespace with gzip inline compression
- `/@kubernetes/namespaces/<namespace>/virtualmachines/<vm_name>.yaml` Virtual machine definition in the namespace
- `/@kubernetes/persistentvolumes/<pv_name>.yaml` Persistent Volume definition
- `/@kubernetes/storageclass/<sc_name>.yaml` Storage Class definition

All supported OpenShift objects will be saved if no filter options are set. You may limit which objects are saved using filtering options described in the `k8sbackupparameters` chapter. By default, if no filter options are set, all supported OpenShift objects will be saved. To see the OpenShift Objects that may be filtered, a listing mode is available. This mode is described in the `k8sobjectlisting` chapter.

Note

When backing up virtual machines, you are only backing up those virtual machines and their dependencies.

Persistent Volume Claim Backup

CommunityEnterprise All Pods in OpenShift are ephemeral and may be destroyed manually or by operations from controllers. Pods do not store data locally because stored data would be destroyed with a pod's life cycle management, so data is saved on Persistent Volumes using Persistent Volume Claim objects to control Volume Space availability.

This brings a new challenge to data backup. Fortunately most of the challenges found here are similar to standard bare metal or virtualized environments. As with bare metal and virtual machine environments, data stored in databases should be protected with dedicated Bacula Enterprise plugins that take advantage of the database backup routines.

See the appropriate Bacula Enterprise plugin documentation for more details on database backups.

On the other hand, most non-database applications store data as simple flat files we can backup as-is without forcing complicated transactions or data consistency procedures. This use case is handled directly with the OpenShift Plugin using a dedicated Bacula Backup Proxy Pod executed in the cluster.

If the container application is more complex, it is possible to execute commands inside the container to quiesce the application:

- before the volume cloning or snapshot
- after the volume cloning or snapshot
- after the backup of the container.

The execution of a command may cause the backup of the container to be terminated due to an issue with the `run.*.failonerror` annotation. You can find detailed description of this feature [here](#).

A Bacula Backup Proxy Pod is a service executed automatically by the OpenShift Plugin which manages secure access to Persistent Volume data for the plugin. It is executed on the OpenShift cluster infrastructure and requires a network connection to the OpenShift Plugin for data exchange on backup and restore operations. No external cluster service like `NodePort`, `LoadBalancer`, `Ingress` or `Host Based Networking` configuration is required to use this feature.

It is also not required to permanently deploy and run this service on the cluster itself as it is executed on demand. The Bacula Backup Proxy Pod does not consume any valuable compute resources outside of the backup window. You can even operate your OpenShift backup solution (Bacula Enterprise service with OpenShift Plugin directly from your on-premises backup infrastructure to backup a public OpenShift cluster (it requires a simple port forwarding firewall rule) or use public backup infrastructure to back up on-premises OpenShift cluster(s). Support for these varied architecture modes is built into the OpenShift Plugin. It is designed to be a *One-Click* solution for OpenShift backups.

Starting from Bacula Enterprise version **12.6.0**, you can back up and restore any PVC data including PVCs not attached to any running OpenShift Pod. This removes a previous limitation in this area.

PVC Backup Modes

CommunityEnterprise To ensure the backup of PVC data, there are three available modes:

- **Standard Mode:** In this mode, the tar program generates a tar pipe from the original data in the PVC, and then the plugin backs it up. a drawback of this method is that if a file is being written while the tar pipe is generating, it may cause issues with the backup and compromise data integrity. On the positive side, this backup mode is slightly faster than the others.
- **Clone Mode:** The plugin instructs the OpenShift cluster to clone a persistent volume, followed by backing up all data from this PVC. However, this backup method lacks data consistency guarantee. The persistent volume CSI driver needs to support Volume Cloning in order to utilize this backup mode.
- **Snapshot Mode:** The plugin initiates a request to the OpenShift Cluster for a snapshot of the persistent volume, followed by a request to create a PVC from the snapshot volume. Subsequently, the plugin proceeds to back up all data, ensuring data consistency. To utilize this backup mode, the persistent volume CSI driver must be compatible with Volume Snapshot.

In terms of data consistency, the order is as follows: Snapshot, Clone and Standard. Nevertheless, not all persistent volumes are capable of supporting snapshot technology. It is crucial to verify the compatibility of this technology. In case Snapshot is not supported, the plugin will seamlessly transition to Clone, and if Volume Cloning is not supported, it will then move to Standard.

PVC Backup Flow

CommunityEnterprise The plugin has the ability to incorporate multiple pathways in order to enhance reader comprehension. They are:

1. Pod and PVC annotations:

- If the parameter **pvcddata** is defined in the Fileset and a Pod has **bacula** annotations referencing PVC backup, the PVCs listed in the Pod annotations will be backed up even if they are not defined in the **pvcddata** parameter.
- The plugin scans all Pods in the namespace specified in the Fileset. If a Pod has the required annotations, the plugin includes the persistent volumes referenced by those annotations, using the corresponding **backup_mode**.
- If a PVC is defined in a Pod annotation, the plugin uses the backup mode specified in the Pod annotation.
- Starting with Bacula Enterprise version **18.1.0**, if a PVC has a **bacula.backup_mode** annotation, this value takes precedence over the backup mode indicated in the Pod annotation.
- If the **pvcddata** parameter is defined in the Fileset but no PVC are specified, the plugin performs backup of all PVCs in the specified namespaces.
- If a persistent volume is included both in a Pod annotation and the Fileset **pvcddata** option, the Pod annotation takes precedence.

2. Backup mode:

- By default, the backup mode is snapshot.
- Backup mode defined in the PVC annotation takes precedence over the backup mode defined in the Pod annotations (since version **18.1.0**). Then, the backup mode defined in the Pod annotation takes precedence over the backup mode defined in the Fileset. This priority is applicable only to the persistent volume claim (PVC) that is annotated in the Pod.

- If a PVC is not compatible with snapshot mode, the plugin attempts the backup using clone mode.
- If the PVC clone contains no data (the backup result of the PVC clone is empty), the plugin attempts to perform the backup using the standard mode, as long as the PVC is not compatible with the clone.
- Snapshot mode is generally the best option for achieving consistent backups.

Note

Starting with Bacula Enterprise version **16.0.14**, the plugin does not back up PVC data if it is in the `Pending` or `Terminating` state.

OpenShift Annotations

CommunityEnterprise This feature allows you to define what volumes (PVC Data) to back up, where and what commands to execute, and how to react to some failures to achieve the best results from data volume backup functionality. You can select which volumes mounted in a Pod you want to backup, select the preferred backup mode for the Pod and PVC, and specify commands to run.

The annotation feature allows administrators of a OpenShift Cluster, or any OpenShift cluster user with enough permissions, to configure a persistent volume to be included in the backup.

The supported annotations are:

bacula/backup.mode:[snapshot|clone|standard]

defines how to handle PVC Data backups. If not defined, the default is `snapshot`.

Note

This annotation is supported on both Pods and PVCs. This type of backup follows the flow described here.

bacula/backup.volumes:<pvcname[,pvcname2...]>

defines which volumes are backed up from this Pod. Multiple PVC names may be selected as a comma-separated list. This annotation is required if you want to back up volumes from this Pod. Any other annotations are optional.

bacula/run.before.job.container.command:<container>/<command>

defines the command to execute in the specific container of the Pod before the volume cloning (or snapshot) and before the Pod's data backup starts. An asterisk (*) as `<container>` name means to execute `<command>` on all containers.

bacula/run.before.job.failjobonerror:[yes|no]

defines whether to fail the job when the exit code of the executed command is not zero. The default is `yes`.

bacula/run.after.job.container.command:<container>/<command>

defines the command to execute in the specific container of the Pod after volume cloning (or snapshot) and after the Pod's data backup completes. An asterisk (*) as `<container>` name means to execute `<command>` on all containers.

bacula/run.after.job.failjobonerror:[yes|no]

defines whether to fail the job when exit code of the executed command is not zero. The default is `no`.

bacula/run.after.snapshot.container.command:<container>/<command>

defines the command to execute in the specific container of the Pod after volume cloning (or snapshot) and just before any data backup starts. An asterisk (*) as <container> name means to execute <command> on all containers. Note that the **snapshot** mode does not perform a volume snapshot but a volume clone.

bacula/run.after.snapshot.failjobonerror:[yes|no]

defines whether to fail the job when exit code of the executed command is not zero. The default is no.

Annotations are an extension to the current PVC Data backup feature available with the **pvcdata=...** plugin parameter as described in `k8sbackupparameters`. This is an independent function which may be used together with the functionality described above, especially since both use the same data archive stream handling with Bacula Backup Pod.

To use this feature, configure the annotations on the selected Pods and ensure that backup for the required OpenShift namespace is properly configured. No changes to the plugin configuration are required.

Note

Even if the PVC is not listed in the parameter **pvcdata**, the backup will always be performed if the pod has pod annotations for PVC backup.

Annotations Examples

CommunityEnterprise Here are a few examples demonstrating how to set up Bacula annotations within OpenShift Pods and PVCs.

When dealing with OpenShift deployments, the annotations must be placed in the Pod template's specification, or `.template.spec` field:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
    tier: mysql
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
      annotations:
        bacula/backup.mode: standard
        bacula/backup.volumes: mysql-pv-claim
    spec:
```

(continues on next page)

(continued from previous page)

```
containers:
  - image: mysql:5.6
    name: mysql
    env:
      - name: MYSQL_ROOT_PASSWORD
        value: changeme
    ports:
      - containerPort: 3306
        name: mysql
    volumeMounts:
      - name: mysql-persistent-storage
        mountPath: /var/lib/mysql
volumes:
  - name: mysql-persistent-storage
    persistentVolumeClaim:
      claimName: mysql-pv-claim
```

In the example below, a simple Linux command, `sync`, is used to synchronize cached writes to persistent storage before volume cloning.

```
apiVersion: v1
kind: Pod
metadata:
  name: app1
  namespace: default
  annotations:
    bacula/backup.mode: clone
    bacula/run.before.job.container.command: "*/sync -f /data1; sync -f /
↪data2"
    bacula/run.before.job.failjobonerror: "no"
    bacula/backup.volumes: "pvc1,pvc2"
spec:
  containers:
    - image: ubuntu:latest
      name: test-container
      volumeMounts:
        - name: pvc1
          mountPath: /data1
        - name: pvc2
          mountPath: /data2
  volumes:
    - name: pvc1
      persistentVolumeClaim:
        claimName: pvc1
    - name: pvc2
      persistentVolumeClaim:
        claimName: pvc2
```

In the example below, PostgreSQL's data-file quiescing is used to ensure a consistent backup when using volume cloning.

Note

A complete PostgreSQL backup solution requires additional configuration and preparation. Those steps are omitted here for clarity.

The initial directive (`run.before.job.container.command`) prevents updates to the database files, while the subsequent instruction (`run.after.snapshot.container.command`) restores normal database functionality once the PVC volume cloning process is completed.

```
apiVersion: v1
kind: Pod
metadata:
  name: postgresql13
  namespace: default
  annotations:
    bacula/backup.mode: standard
    bacula/run.before.job.container.command: "*/bin/startpgsqlbackup.sh"
    bacula/run.after.snapshot.container.command: "*/bin/stoppgsqlbackup.sh"
    bacula/run.after.snapshot.failjobonerror: "yes"
    bacula/backup.volumes: "pgsql"
spec:
  containers:
  - image: postgresql:13
    name: postgresql-server
    volumeMounts:
    - name: pgsq1
      mountPath: /var/lib/pgsql
  volumes:
  - name: pgsq1
    persistentVolumeClaim:
      claimName: pgsq1-volume
```

In the following example, `pvc1`, is backed up using clone mode, while `pvc2` is backed up using the standard backup method.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc2
  annotations:
    bacula/backup.mode: standard
```

(continues on next page)

(continued from previous page)

```
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: appl
  namespace: default
  annotations:
    bacula/backup.mode: clone
    bacula/backup.volumes: "pvc1,pvc2"
spec:
  containers:
    - image: ubuntu:latest
      name: test-container
      volumeMounts:
        - name: pvc1
          mountPath: /data1
        - name: pvc2
          mountPath: /data2
  volumes:
    - name: pvc1
      persistentVolumeClaim:
        claimName: pvc1
    - name: pvc2
      persistentVolumeClaim:
        claimName: pvc2
```

To ensure that a PVC is always backed up using a specific mode, specify the mode in the PVC annotations, for example:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-with-annotations
  annotations:
    bacula/backup.mode: clone
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

All PVCs defined with this storage class will perform backups using snapshots as per this storage definition.

```
apiVersion: storage.k8s.io/v1
```

(continues on next page)

(continued from previous page)

```
kind: StorageClass
metadata:
  name: csi-hostpath-sc
provisioner: hostpath.csi.k8s.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: plugin-test-persistent-volume-claim-csi
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-hostpath-sc
```

Run Container Command Annotation

CommunityEnterprise All flavors of the Run Container Command parameters are remotely executed using the OpenShift Pod remote execution API. Every command is prepared to execute with a standard Linux shell `/bin/sh`. This requires that a container image has to have the specified shell available. Using command shell execution gives flexibility to command execution or even allows for preparation of small scripts without additional container image customization.

CSI Volume Features Support

CommunityEnterprise Starting with Bacula Enterprise 12.6.0, support for OpenShift CSI Volume Cloning was introduced, along with other enhancements. Later, starting with Bacula Enterprise 16.0.8, OpenShift CSI Volume Snapshot support was added. From that version onward, the plugin automatically detects whether it can use this functionality. As of 16.0.13, users can choose the backup mode that best fits their needs while maintaining compatibility.

Starting from Bacula Enterprise version **16.0.8**, you can use CSI Volume Cloning or CSI Volume Snapshot to obtain a consistent data view of selected Volumes.

Additionally, you can configure remote command execution in a selected Container of the Pod. This command execution can be configured just before or after a Pod backup and just after volume snapshot/clone creation. More details in [here](#).

CSI drivers may or may not implement volume cloning and/or snapshot functionality. For reference, see [CSI Volume Cloning](#) and [Volume Snapshots](#).

The main distinction between CSI Volume Cloning and CSI Volume Snapshot is that Volume Cloning creates an exact duplicate of the specified volume, meanwhile, the Volume Snapshot represents a point-in-time copy of a volume. Therefore, a snapshot-based backup offers greater consistency.

When performing persistent volume backups, the plugin uses the volume clone and snapshot APIs. Because snapshotting generally provides a higher level of consistency, the plugin uses snapshots by default whenever the persistent volume's CSI driver supports them.

Etcd Backup

CommunityEnterprise Starting with Bacula Enterprise version **18.2.0**, the plugin includes support for backing up etcd. It also introduces new parameters (prefixed with `etcd`) specifically for etcd backups. Both single-node etcd deployments and high-availability (HA) configurations are supported.

What is Etcd?

Etcd is a reliable and highly available key-value store that serves as the backend for all cluster data in OpenShift. Its main role in a OpenShift cluster is to provide central storage for the cluster's state, tracking all resources like Pods, Services, configuration objects and more.

When you make changes to the configuration in OpenShift, such as deploying a new service or removing a Pod, these updates are recorded in etcd. If etcd fails or is lost, OpenShift loses all information regarding the current cluster state and configuration. Consequently, if etcd becomes corrupted, the OpenShift cluster may become unusable. In a production environment, this could lead to the loss of essential applications and data, significantly affecting service availability and business continuity.

Why is Backing Up Important?

Etcd acts as the “source of truth” for your OpenShift cluster. Regular backups provide:

- **Disaster Recovery:** Restore the cluster after data loss, corruption, or a major cluster failure.
- **Business Continuity:** Reduce data loss and recovery time, ensuring that services remain operational.

Accessing Etcd Resources

Typically, etcd resides in the `kube-system` namespace called. You can access this database remotely using etcd certificates.

To obtain these certificates, you need to know their location on one of the etcd nodes (nodes with role etcd). You can find this information by checking the etcd Pod YAML file to see where the certificates are mounted on the host. You can then access it remotely to retrieve the certificates.

One way to get the Pods with etcd service is:

```
kubectl -n kube-system get pods | grep etcd
etcd-rke2-master01-bck          1/1    Running    2
↪ 33d
etcd-rke2-master02-bck          1/1    Running    1
↪ 33d
etcd-rke2-master03-bck          1/1    Running    1
↪ 33d
```

Next, identify which configuration file is used by etcd:

```
kubectl -n kube-system get pods etcd-rke2-master01-back -o yaml | grep "\-file"
↪ "
- --config-file=/var/lib/rancher/rke2/server/db/etcd/config
```

Now you can access an etcd node or an etcd Pod, and locate the certificate paths referenced in that configuration file:

```

root@rke2-master01-bck:~# cat /var/lib/rancher/rke2/server/db/etcd/config |
↵grep file
cert-file: /var/lib/rancher/rke2/server/tls/etcd/server-client.crt
key-file: /var/lib/rancher/rke2/server/tls/etcd/server-client.key
trusted-ca-file: /var/lib/rancher/rke2/server/tls/etcd/server-ca.crt
cert-file: /var/lib/rancher/rke2/server/tls/etcd/peer-server-client.crt
key-file: /var/lib/rancher/rke2/server/tls/etcd/peer-server-client.key
trusted-ca-file: /var/lib/rancher/rke2/server/tls/etcd/peer-ca.crt

```

The important values of keys are: **cert-file: server-client.crt**, **key-file:server-client.key** and **trusted-ca-file:server-ca.crt** . These files allow you to access the etcd database. In this example, they are:

```

cert-file: /var/lib/rancher/rke2/server/tls/etcd/server-client.crt
key-file: /var/lib/rancher/rke2/server/tls/etcd/server-client.key
trusted-ca-file: /var/lib/rancher/rke2/server/tls/etcd/server-ca.crt

```

Import these files to your host and store them in a OpenShift Secret:

Note

By default, the certificates expire after one year.

```

root@kubeadm-master01-bck:~# kubectl cp <etcd-pod-name>:/path/to/server-ca.
↵crt ./server-ca.crt -n kube-system
root@kubeadm-master01-bck:~# kubectl cp <etcd-pod-name>:/path/to/server-
↵client.crt ./server-client.crt -n kube-system
root@kubeadm-master01-bck:~# kubectl cp <etcd-pod-name>:/path/to/server-
↵client.key ./server-client.key -n kube-system

```

The recommended method for importing these certificates into the OpenShift Plugin is to create a OpenShift Secret in the namespace where the etcd Pod is located. You can do this by following these steps:

```

root@kubeadm-master01-bck:~# kubectl create secret generic etcd-certificates \
↵--from-file=server-ca.crt=./server-ca.crt \
↵--from-file=server-client.crt=./server-client.crt \
↵--from-file=server-client.key=./server-client.key \
↵-n kube-system

```

Note

The key names in the Secret must be: `server-ca.crt`, `server-client.crt` and `server-client.key`.

Note

The Secret must be created in the `kube-system` namespace.

The final step is to add the `etcd_secret` parameter to the Bacula Fileset.

Alternatively, you can store the certificates in the Bacula File Daemon and include the parameters `etcd_ca_cert`, `etcd_tls_cert` and `etcd_tls_key` in the Bacula Fileset.

You must also set `etcd_endpoints` in the Fileset. You can set this parameter using the following command:

```
root@rke2-master01-bck:~# kubectl get pod -n kube-system -o wide | grep etcd
etcd-rke2-master01-bck          1/1      Running   0
↪ 19h    10.0.97.201  rke2-master01-bck <none>   <none>
etcd-rke2-master02-bck          1/1      Running   0
↪ 19h    10.0.97.202  rke2-master02-bck <none>   <none>
etcd-rke2-master03-bck          1/1      Running   0
↪ 19h    10.0.97.203  rke2-master03-bck <none>   <none>
```

Based on this output, you can set: `etcd_endpoints=https://10.0.97.201:2379,https://10.0.97.202:2379,https://10.0.97.203:2379`. You can also use hostnames, but ensure DNS resolution is working correctly.

In the future, the plugin will create this OpenShift Secret with a `.query` command.

4.2 Restore

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The OpenShift Plugin provides two targets for restore operations:

- Restore to an OpenShift cluster
- Restore to a local directory

Restore Etcd

CommunityEnterprise To restore OpenShift etcd, you must restore the etcd snapshot from a previous backup and place it in a local directory. You need to mark the file: `@kubernetes/namespaces/<etcd-namespace>/etcd/snaposhot-<day>-<month>-<year>.db`.

For detailed instructions on performing a local restore, refer to `RestoreToLocalDirectory`.

Once the restore is complete, transfer the restored snapshot to the master node in your OpenShift cluster. The exact steps depend on your OpenShift distribution, so consult the official documentation for your distribution.

We have provided a straightforward example using RKE2 in `RestoreEtcdRke2Example`.

Restore to OpenShift Cluster

CommunityEnterprise To use this restore method, the `where=/` parameter of a Bacula restore command is used. You can select any supported OpenShift object to restore, or perform a batch restore of entire whole namespace or even multiple namespaces.

If you select a single object to restore, it is restored as is without any dependent objects. In most cases (Config Maps, Secrets, Services, etc.), this is fine and restore succeeds. When you restore compound objects (Pods, Deployments, Ingress, etc.), the restored objects may not become ready unless all dependencies are also restored. In these cases, ensure you select all required objects.

Furthermore, it is important to note that certain containers such as Postgresql, MySQL, etc., require PVC data. In the event that these containers do not locate data within PVC, they generate new ones. Starting with Bacula Enterprise version **16.0.14**, there is no need for concern as the plugin handles the restore seamlessly.

In OpenShift, a successful object restore does not necessarily mean the service comes online immediately. Additional monitoring and troubleshooting may be required. For example:

- *Container image is unavailable.*
- *Volume Claim cannot provision new volume.*
- *Untrusted Image Repository.*
- *Infrastructure limits exceeded, i.e. a number of Pods or Memory allocations.*
- *etc...*

All of the above issues must be resolved by the OpenShift administrator. Once resolved, the objects should come online automatically. If not, you may need to repeat the restore to redeploy the object configuration.

The OpenShift Plugin does not wait for an object to come online during restore. It checks the status of the object creation or replace operation and reports any errors in the job log. The only exception to this is PVC Data restore, when the OpenShift Plugin waits for a successful archive data restore. Starting with Bacula Enterprise version **16.0.14**, this operation is always executed before creating compound components (Pods, Deployments, etc.).

Restore to Local Directory

CommunityEnterprise To use this mode, the **where=/some/path** Bacula **restore** parameter is set to a full path on a server where the Bacula File Daemon and OpenShift Plugin are installed. If the path does not exist, it will be created by the OpenShift Plugin. With this restore mode, you can restore any saved OpenShift Object including PVC Data archive file to a location on disk.

Restore PVC Data

CommunityEnterprise This section describes the functionality and requirements related to pvcdata restore.

Local Directory Restore

The procedure to restore a PVC Data archive file to a local directory is essentially the same as restoring the OpenShift Object configuration file as described in RestoreToLocalDirectory. However, output transformation is unavailable and ignored when restoring PVC data.

If the PVC uses a filesystem volume mode, the restore creates a tar archive file you can manually inspect and use. If the PVC uses a block volume mode, the data creates a raw archive file (or a compressed file if compression was enabled during backup), which you can manually mount and use on your system.

Restore to PVC

This procedure is similar to the one described in PVC Data backup and uses the same Bacula Backup Proxy Pod image. During restore, the plugin uses the same endpoint configuration parameters so it is not necessary to configure it again. If the endpoint parameters have changed, you can update them using Bacula plugin restore options modification as in the example below:

```
*restore select all done where=/
(...)
OK to run? (yes/mod/no): mod
Parameters to modify:
  1: Level
  2: Storage
  3: Job
  4: FileSet
  5: Restore Client
  6: When
  7: Priority
  8: Bootstrap
  9: Where
 10: File Relocation
 11: Replace
 12: JobId
 13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : kubernetes: namespace=plugintest pvcdat
↵pluginhost=example.com
Plugin Restore Options
config:          *None*          (*None*)
host:            *None*          (*None*)
token:           *None*          (*None*)
verify_ssl:     *None*          (True)
ssl_ca_cert:    *None*          (*None*)
outputformat:   *None*          (RAW)
fdaddress:      *None*          (*FDAddress*)
fdport:         *None*          (9104)
pluginhost:     *None*          (*FDAddress*)
pluginport:     *None*          (9104)
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
  1: config (K8S config file)
  2: host (K8S API server URL/Host)
  3: token (K8S Bearertoken)
  4: verify_ssl (K8S API server cert verification)
  5: ssl_ca_cert (Custom CA Certs file to use)
  6: outputformat (Output format when saving to file (JSON, YAML))
  7: fdaddress (The address for listen to incoming backup pod data)
  8: fdport (The port for opening socket for listen)
  9: pluginhost (The endpoint address for backup pod to connect)
 10: pluginport (The endpoint port to connect)
Select parameter to modify (1-10): 9
Please enter a value for pluginhost: newbackup.example.com
Plugin Restore Options
```

(continues on next page)

(continued from previous page)

```
config:          *None*          (*None*)
host:            *None*          (*None*)
token:          *None*          (*None*)
verify_ssl:     *None*          (True)
ssl_ca_cert:    *None*          (*None*)
outputformat:   *None*          (RAW)
fdaddress:      *None*          (*FDAddress*)
fdport:         *None*          (9104)
pluginhost:     newbackup.example.com (*FDAddress*)
pluginport:     *None*          (9104)
Use above plugin configuration? (yes/mod/no): yes
```

When restoring PVC data, the plugin restores all data from the backup archive for the selected Persistent Volume Claim. Existing data is overwritten regardless of the `Repl`ace setting. Be aware of this behavior; it may change in a future release.

Restore Examples

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

Restore to Local Directory Example

CommunityEnterprise It is possible to restore any OpenShift Object(s) to file without loading them into a cluster. To do so, the **where** restore option should point to the local directory:

```
* restore where=/tmp/bacula/restores
...
$ cd /@kubernetes/namespaces/
cwd is: /@kubernetes/namespaces/
$ ls
bacula/
cattle-system/
default/
graphite/
ingress/
plugintest/
$ add plugintest
25 files marked.
$ done
Bootstrap records written to /opt/bacula/working/bacula-devel-dir.restore.2.
↵bsr
```

The Job will require the following (*=>InChanger):

Volume(s)	Storage(s)	SD Device(s)
-----------	------------	--------------

=====

(continues on next page)

(continued from previous page)

```
Vol005                File1                FileChgr1

Volumes marked with "*" are in the Autochanger.

25 files selected to be restored.

Run Restore job
JobName:      RestoreFiles
Bootstrap:    /opt/bacula/working/bacula-devel-dir.restore.2.bsr
Where:        /tmp/bacula/restores
Replace:      Always
FileSet:      Full Set
Backup Client: bacula-devel-fd
Restore Client: bacula-devel-fd
Storage:      File1
When:         2019-09-30 12:58:16
Catalog:      MyCatalog
Priority:      10
Plugin Options: *None*
OK to run? (yes/mod/no): mod
Parameters to modify:
  1: Level
  2: Storage
  3: Job
  4: FileSet
  5: Restore Client
  6: When
  7: Priority
  8: Bootstrap
  9: Where
 10: File Relocation
 11: Replace
 12: JobId
 13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : kubernetes: config=/home/radekk/.kube/config debug=1
Plugin Restore Options
config:        *None*                (*None*)
host:          *None*                (*None*)
token:         *None*                (*None*)
verify_ssl:    *None*                (True)
ssl_ca_cert:   *None*                (*None*)
outputformat:  *None*                (RAW)
fdaddress:     *None*                (*FDAddress*)
fdport:        *None*                (9104)
pluginhost:    *None*                (*FDAddress*)
pluginport:    *None*                (9104)
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
  1: config (K8S config file)
```

(continues on next page)

(continued from previous page)

```
2: host (K8S API server URL/Host)
3: token (K8S Bearertoken)
4: verify_ssl (K8S API server cert verification)
5: ssl_ca_cert (Custom CA Certs file to use)
6: outputformat (Output format when saving to file (JSON, YAML))
7: fdaddress (The address for listen to incoming backup pod data)
8: fdport (The port for opening socket for listen)
9: pluginhost (The endpoint address for backup pod to connect)
10: pluginport (The endpoint port to connect)
Select parameter to modify (1-8): 8
Please enter a value for outputformat: JSON
Plugin Restore Options
config:          *None*          (*None*)
host:            *None*          (*None*)
token:           *None*          (*None*)
verify_ssl:      *None*          (True)
ssl_ca_cert:     *None*          (*None*)
outputformat:    *None*          (RAW)
fdaddress:       *None*          (*FDAddress*)
fdport:          *None*          (9104)
pluginhost:      *None*          (*FDAddress*)
pluginport:      *None*          (9104)
Use above plugin configuration? (yes/mod/no): yes
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/bacula-devel-dir.restore.2.bsr
Where:           /tmp/bacula/restores
Replace:         Always
FileSet:         Full Set
Backup Client:   bacula-devel-fd
Restore Client:  bacula-devel-fd
Storage:         File1
When:            2019-09-30 12:58:16
Catalog:         MyCatalog
Priority:         10
Plugin Options:  User specified
OK to run? (yes/mod/no):
Job queued. JobId=1085
```

Output format conversion at restore time will format all data in a human readable format. You can find an example of this restore below.

```
# cat /tmp/bacula/restores/namespaces/pluginintest/pluginintest.json
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "annotations": {
      "field.cattle.io/projectId": "c-hb9ls:p-bm6cw",
      "lifecycle.cattle.io/create.namespace-auth": "true"
    },
    "cluster_name": null,
```

(continues on next page)

(continued from previous page)

```
"creation_timestamp": "2019-09-25T16:31:03",
"deletion_grace_period_seconds": null,
"deletion_timestamp": null,
"finalizers": [
  "controller.cattle.io/namespace-auth"
],
"generate_name": null,
"generation": null,
"initializers": null,
"labels": {
  "field.cattle.io/projectId": "p-bm6cw"
},
"name": "plugintest",
"namespace": null,
"owner_references": null,
"resource_version": "11622",
"self_link": "/api/v1/namespaces/plugintest",
"uid": "dd873930-dfb1-11e9-aad0-022014368e80"
},
"spec": {
  "finalizers": [
    "kubernetes"
  ]
},
"status": {
  "phase": "Active"
}
}
```

The supported output transformations are: JSON and YAML.

Restore to OpenShift Cluster Examples

[CommunityEnterprise](#) To restore OpenShift objects to a OpenShift cluster, the administrator should execute the restore command and specify the **where** parameter as in this example:

```
* restore where=/
```

and then set any other required restore plugin parameters for the restore.

```
*restore jobid=1 where=/
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
You have selected the following JobId: 1

Building directory tree for JobId(s) 1 ...
↪+++++
443 files inserted into the tree.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
```

(continues on next page)

(continued from previous page)

```
Enter "done" to leave this mode.

cwd is: /
$ cd @kubernetes/namespaces/testing-ns-0010-1/configmaps
cwd is: /@kubernetes/namespaces/testing-ns-0010-1/configmaps/
$ ls
kube-root-ca.crt.yaml
test-n1-configmap-0010-1.yaml
$ mark *
2 files marked.
$ done
Bootstrap records written to /opt/bacula/working/bacula-devel-dir.restore.46.
↪bsr
The Job will require the following (*=>InChanger):
Volume(s)                Storage(s)                SD Device(s)
=====
Vol-0001                  File1                      FileStorage1

Volumes marked with "*" are in the Autochanger.

2 files selected to be restored.

Automatically selected Client: bacula-devel-fd
Using Catalog "MyCatalog"
Run Restore job
JobName:      RestoreFiles
Bootstrap:    /home/fmgb/bee/projects/bacula-bee/regress/working/bacula-
↪devel-dir.restore.46.bsr
Where:        /
Replace:      Always
FileSet:      Full Set
Backup Client: bacula-devel-fd
Restore Client: bacula-devel-fd
Storage:      File1
When:         2024-11-29 13:34:48
Catalog:      MyCatalog
Priority:      10
Plugin Options: *None*
OK to run? (Yes/mod/no): mod
Parameters to modify:
  1: Level
  2: Storage
  3: Job
  4: FileSet
  5: Restore Client
  6: When
  7: Priority
  8: Bootstrap
  9: Where
 10: File Relocation
```

(continues on next page)

(continued from previous page)

```
11: Replace
12: JobId
13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : kubernetes: debug=1 config='/opt/bacula/etc/k8s/
↪config' fdkeyfile=/opt/bacula/etc/k8s/snakeoil.key fdcertfile=/opt/bacula/
↪etc/k8s/snakeoil.pem namespace=testing-ns-0010-1
Plugin Restore Options
Option                               Current Value           Default Value
config:                              *None*                 (*None*)
host:                                 *None*                 (*None*)
token:                                *None*                 (*None*)
verify_ssl:                          *None*                 (True)
ssl_ca_cert:                         *None*                 (*None*)
outputformat:                        *None*                 (RAW)
fdaddress:                           *None*                 (*FDAddress*)
fdport:                              *None*                 (9104)
pluginhost:                          *None*                 (*FDAddress*)
pluginport:                          *None*                 (9104)
new_namespace:                       *None*                 (*None*)
new_name:                             *None*                 (*None*)
new_storageclass:                   *None*                 (*None*)
Use above plugin configuration? (Yes/mod/no): mod
You have the following choices:
 1: config (K8S config file)
 2: host (K8S API server URL/Host)
 3: token (K8S Bearertoken)
 4: verify_ssl (K8S API server cert verification)
 5: ssl_ca_cert (Custom CA Certs file to use)
 6: outputformat (Output format when saving to file (JSON, YAML))
 7: fdaddress (The address for listen to incoming backup pod data)
 8: fdport (The port for opening socket for listen)
 9: pluginhost (The endpoint address for backup pod to connect)
10: pluginport (The endpoint port to connect)
11: new_namespace (The namespace where the resources will be restored)
12: new_name (The new name of restored resource)
13: new_storageclass (The storageclass that will be applied in pvc,
↪restores)
Select parameter to modify (1-13): 1
Please enter a value for config: /root/.kube/config
Plugin Restore Options
Option                               Current Value           Default Value
config:                              /root/.kube/config     (*None*)
host:                                 *None*                 (*None*)
token:                                *None*                 (*None*)
verify_ssl:                          *None*                 (True)
ssl_ca_cert:                         *None*                 (*None*)
outputformat:                        *None*                 (RAW)
fdaddress:                           *None*                 (*FDAddress*)
fdport:                              *None*                 (9104)
pluginhost:                          *None*                 (*FDAddress*)
pluginport:                          *None*                 (9104)
```

(continues on next page)

(continued from previous page)

```
new_namespace:          *None*          (*None*)
new_name:               *None*          (*None*)
new_storageclass:      *None*          (*None*)
Use above plugin configuration? (Yes/mod/no): yes
Run Restore job
JobName:                RestoreFiles
Bootstrap:             /home/fmgb/bee/projects/bacula-bee/regress/working/bacula-
↳ devel-dir.restore.46.bsr
Where:                 /
Replace:               Always
FileSet:               Full Set
Backup Client:         bacula-devel-fd
Restore Client:        bacula-devel-fd
Storage:               File1
When:                  2024-11-29 13:34:48
Catalog:               MyCatalog
Priority:               10
Plugin Options:        User specified
OK to run? (Yes/mod/no):
Job queued. JobId=1084
```

The plugin does not pause for OpenShift objects to be ready and online like `kubectl` or the `oc` commands do.

Restore to OpenShift Cluster Overwriting Values

To restore OpenShift objects to a OpenShift cluster overriding values such as namespace, name or even StorageClass (the Persistent Volume Claim only), the administrator should execute the restore command and modify the values in Plugin Options menu, as in this example:

```
* restore jobid=1 where=/
...
Select parameter to modify (1-13): 13
Automatically selected : kubernetes: debug=1 config='/opt/bacula/etc/k8s/
↳ config'
fdkeyfile=/opt/bacula/etc/k8s/snakeoil.key fdcertfile=/opt/bacula/etc/k8s/
↳ snakeoil.pem
namespace=testing-ns-0010-1
Plugin Restore Options
Option                Current Value          Default Value
config:               *None*                (*None*)
host:                 *None*                (*None*)
token:                *None*                (*None*)
verify_ssl:           *None*                (True)
ssl_ca_cert:          *None*                (*None*)
outputformat:         *None*                (RAW)
fdaddress:            *None*                (*FDAddress*)
fdport:               *None*                (9104)
pluginhost:           *None*                (*FDAddress*)
pluginport:           *None*                (9104)
new_namespace:        *None*                (*None*)
new_name:              *None*                (*None*)
```

(continues on next page)

(continued from previous page)

```
new_storageclass:          *None*          (*None*)
debug:                    *None*          (*None*)
Use above plugin configuration? (Yes/mod/no):
...
```

Frequently Asked Questions

1. What happens if you try to perform a restore of a OpenShift resource in another namespace that does not exist?

The plugin check if the target namespace exists or not. If it does not exist, the plugin creates a new one.

2. What happens if you try to restore several OpenShift resource with a different name?

The plugin will restore the resources with the same name. If they are the same type, the restore fails.

3. What happens if you try to restore a Persistent Volume Claim to a different StorageClass that does not exist?

The restore fails.

4. What happens if you try to restore a Persistent Volume Claim to a different StorageClass with a different filesystem?

This is supported. The data of Persistent Volume Claim is restored normally.

4.3 Resource Listing

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The Bacula Enterprise OpenShift Plugin supports the “plugin listing” feature of Bacula Enterprise 8.x or newer. This mode allows the plugin to display some useful information about available OpenShift resources such as:

- List of OpenShift namespaces
- List of OpenShift Persistent Volumes
- List of OpenShift storage class resources

The feature uses the special **.ls** command with a **plugin=<plugin>** parameter.

The command requires the following parameters to be set:

client=<client>

A Bacula Client name with the OpenShift plugin installed.

plugin=<plugin>

A plugin name, which would be **openshift**: in this case, with optional plugin parameters as described in section Generic Plugin Parameters.

path=<path>

An object path to display.

The supported values for a **path=<path>** parameter are:

/

to display Object types available to list

namespaces

to display a list of Namespaces

persistentvolumes

to display a list of Persistent Volumes

storageclass

to display a list of Storage Class Resources

namespaces/<name>/pvcdata

to display all available Persistent Volume Claims available in Namespace <name>

To display available Object types, follow the following command example:

```
*.ls plugin=openshift: client=openshift-fd path=/
Connecting to Client openshift-fd at localhost:9102
drwxr-x---  1 root    root 2018-09-28 14:32:20 /namespaces
drwxr-x---  1 root    root 2018-09-28 14:32:20 /persistentvolumes
drwxr-x---  1 root    root 2018-09-28 14:32:20 /storageclass
2000 OK estimate files=2 bytes=0
```

To display the list of all available OpenShift namespaces, the following command example can be used:

```
*.ls plugin=openshift: client=openshift-fd path=namespaces
Connecting to Client openshift-fd at localhost:9102
drwxr-xr-x  1 root    root 2019-09-25 16:39:56 /namespaces/default
drwxr-xr-x  1 root    root 2019-09-25 16:39:56 /namespaces/kube-public
drwxr-xr-x  1 root    root 2019-09-25 16:39:56 /namespaces/kube-system
drwxr-xr-x  1 root    root 2019-09-25 16:46:19 /namespaces/cattle-system
drwxr-xr-x  1 root    root 2019-09-27 13:04:01 /namespaces/plugintest
2000 OK estimate files=5 bytes=0
```

To display the list of available Persistent Volume Claims which could be used for PVC Data archive feature selection, you can use the following example command for the mysql namespace:

```
*.ls client=openshift-fd plugin="openshift:" path=/namespaces/mysql/pvcdata
Connecting to Client openshift-fd at openshift:9102
-rw-r-----  1 root    root 2019-10-16 14:29:38 /namespaces/mysql/pvcdata/
↪mysql-mysql
2000 OK estimate files=1 bytes=0
```

To display the list of all available Persistent Volumes, the following command example can be used:

```
*.ls plugin=openshift: client=openshift-fd path=persistentvolumes
Connecting to Client openshift-fd at localhost:9102
-rw-r-----  1073741824 2019-09-25 /persistentvolumes/pvc-bfaebd0d-dfad-11e9-
↪a2cc-42010a8e0174
-rw-r-----  1073741824 2019-09-25 /persistentvolumes/pvc-b1a49497-dfad-11e9-
↪a2cc-42010a8e0174
```

(continues on next page)

(continued from previous page)

```
-rw-r----- 1073741824 2019-09-25 /persistentvolumes/pvc-949cb638-dfad-11e9-
↪a2cc-42010a8e0174
-rw-r----- 1073741824 2019-09-25 /persistentvolumes/pvc-9313388c-dfad-11e9-
↪a2cc-42010a8e0174
-rw-r----- 10737418240 2019-09-24 /persistentvolumes/myvolume
2000 OK estimate files=5 bytes=15,032,385,536
```

The volume lists display a Volume Storage size which does not reflect the actual configuration size during backup.

To display the list of all defined Storage Class Resources, the following command example can be used:

```
*.ls plugin=openshift: client=openshift-fd path=storageclass
Connecting to Client openshift-fd at openshift:9102
-rw-r----- 1024 2020-07-27 13:39:48 /storageclass/local-storage
-rw-r----- 1024 2020-07-23 16:14:13 /storageclass/default-postgresql-1
-rw-r----- 1024 2020-07-24 11:47:02 /storageclass/local-storage-default
-rw-r----- 1024 2020-07-23 12:00:02 /storageclass/standard
2000 OK estimate files=4 bytes=4,096
```

5 OpenShift: Autonomous Actions

CommunityEnterprise This section provides a detailed explanation of the decisions the plugin makes automatically based on the environment variables and context in which it runs.

5.1 Auto-Skip of PVCs

The plugin automatically skips PVCs that are in **Pending** or **Terminating** state. It also ignores PVCs associated with other Bacula jobs.

5.2 Auto-Detection of Filesystem Format

Before starting a backup in snapshot or cloning mode, **when the Persistent Volume Claim (PVC) is compatible with snapshot or cloning**, the plugin checks whether the file systems of the Persistent Volume (PV) and the StorageClass match. If it cannot detect the file system on either resource or if they differ, the plugin records a warning in the job log but does not change the selected backup mode. Then, it proceeds with the originally chosen mode, though the backup may be highly prone to failure if the file systems are incompatible.

5.3 Regeneration of XFS UUID

In Snapshot and Clone backup modes, a full clone of the Persistent Volume Claim is created, including the volume's UUID. This can cause issues when using file systems that are sensitive to duplicate UUIDs, such as XFS.

When this situation is detected, the plugin automatically performs the necessary actions to generate a new UUID for the cloned device, preventing potential conflicts.

Note

These actions are only triggered if the file system of the defined StorageClass is XFS and the `longhorn_url` parameter is defined.

5.4 Direct Selection of Backup StorageClass

In certain situations, it is advisable to perform the backup using a StorageClass different from the one defined for the original PVC.

If the plugin detects StorageClasses named `bacula-backup-xf`s and `bacula-backup-ext4` in the environment, it will automatically use them. The selection depends on the file system used by the Persistent Volume.

Note

This is only useful when performing the backup in Clone or Snapshot mode.

5.5 Direct Node Selection when Attaching the Longhorn Cloned PVC

During backup operations Clone or Snapshot mode, if the provisioner is Longhorn, the plugin automatically detects the PVC's access mode and determines on which node the cloned volume must not be attached.

If the access mode is RWO, the plugin avoids mounting the cloned volume on the same node as the PVC being backed up.

For other access modes, the plugin avoids mounting the cloned volume on the same node as the `share-manager` pod, which Longhorn uses to enable multi-access.

5.6 Using a StorageClass with Binding Mode “WaitForFirstConsumer”

When cloning a PVC in Clone or Snapshot mode, the StorageClass used for the backup may be configured with the binding mode “WaitForFirstConsumer” mode.

In this case, the plugin detects this situation and launches a simple Job to perform the binding. Once the PVC is available for backup, the plugin launches the Bacula backup pod.

5.7 Retry Standard Backup if Clone Backup Fails

When a PVC is cloned during a clone-based backup, the provisioner of the StorageClass may not be compatible and might create a clone without the content of the original PVC. During this operation, no error is reported.

To detect this situation, the plugin scans the cloned PVC after backup to verify that files are present. If no files are found, the plugin automatically retries the backup using the Standard backup method.

6 OpenShift: Limitations

CommunityEnterprise

- Only full level backups are possible. This is a OpenShift limitation.
- The standard backup is not compatible with PVC with block volume.
- You can restore `etcd` only by using a local restore.
- The `restart` command has limitations with plugins, as it initiates the Job from scratch rather than continuing it. Bacula determines whether a Job is restarted or continued, but using the `restart` command will result in a new Job.

- PersistentVolumeClaims intended to be backed up via snapshot mode must belong to a Storage-Class that uses the same type of filesystem format. It is worth noting that the default filesystem in OpenShift is 'ext4'.
- Static PersistentVolumeClaims can only use the standard backup mode.
- The plugin uses certain prefixes in the name of the PersistentVolumeClaims(PVCs) to identify the PVCs it has created itself. It is recommended not to use any of these prefixes to avoid issues. The prefixes are:
 - bacula-pvcfs
 - bacula-pvccclone

7 OpenShift: Troubleshooting

[CommunityEnterprise](#) In this section we describe common problems and ways to solve them.

7.1 Error: kubernetes: incluster error: Service host/port is not set

You have set the **incluster** parameter in your Job but you have the following error:

```
Error: kubernetes: incluster error: Service host/port is not set.
```

This means you are running the Bacula File Daemon and OpenShift plugin not in a Pod, or OpenShift does not provide default service access in your installation. In the latter case you should use a standard OpenShift access method in a prepared kubeconfig file.