# Kubernetes Plugin

## Bacula Systems Documentation

# Contents

# Contents

---

Containers provide a lightweight system-level virtualization solution with minimal overhead, as applications within virtual partitions utilize the operating system's standard system call interface without requiring emulation or running in a separate virtual machine. Kubernetes manages a set of containers to create a flexible execution environment for various applications and services

The Bacula Enterprise Kubernetes Plugin is designed to securely store all critical Kubernetes objects and meta-data that are essential for the functioning of the application or service. This includes the following name-spaced objects:

- Config Map
- Daemon Set
- Deployment
- Endpoint
- Ingress (From Bacula Enterprise version **16.0.14**)
- Limit Range
- Pod
- Persistent Volume Claim
- Pod Template
- Replica Set
- Replication Controller
- Resource Quota
- Secret
- Service
- Service Account
- Stateful Set
- PVC Data Archive

**Note:** The PVC Data does not precisely correspond to a Kubernetes Object, but rather serves as a repository of actual data that exists on the chosen PVC.

and non name-spaced objects:

- Namespace
- Persistent Volume
- Storage Class

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

Proper authorization is necessary for users and service accounts to access the server API. This involves going through authentication, authorization, and admission control. In order to effectively back up Kubernetes objects, it is essential to have a user or service account with the appropriate permissions and rights to be authenticated and authorized to access the API server and the objects that need to be backed up.

To ensure successful object configuration backups, it is essential for the user or service account to have the capability to read and list the objects. Additionally, for PVC Data backup, the user or service account must also possess the ability to create and delete pods. This is necessary as the plugin will need to create and delete the Bacula Backup Proxy Pod during the backup process.

See the Kubernetes documentation for more details.

Read more in the sub-chapters:

# 1 Features

The artile aims at presenting the Kubernetes Plugin features.

They are:

- Kubernetes cluster objects configuration backup
- Ability to restore single Kubernetes configuration object
- Ability to restore Kubernetes object configuration to local directory
- Kubernetes Persistent Volumes data backup and restore
- Ability to restore Kubernetes Persistent Volumes data to local directory
- Ability to use Kubernetes CSI driver volume snapshot and cloning features 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 Kubernetes workload backup requirements directly with Pod annotations.
- Cleanup pods and cloned pvcs from old backup jobs that, for any reason, did not finish correctly. This feature is automatic.

**See also:**

Go to:

- *Installation*

Go back to the main *Kubernetes Plugin page*.

# 2 Installation

The Bacula File Daemon and the Kubernetes Plugin can be installed outside of the Kubernetes cluster on a server which has access to the Kubernetes API, or inside a protected cluster in a Container/Pod. The Kubernetes 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 Kubernetes Plugin on a Kubernetes 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
  ...
}
```

Read about the installation and deployment:

### Kubernetes Plugin Installation with Package Manager

Installation of the Bacula Enterprise Kubernetes Plugin is most easily done by adding the Kubernetes repository from your Bacula Enterprise subscription to your Linux distribution's package manager configuration.

An example would be `/etc/apt/sources.list.d/bacula.list` for Debian-based Linux distributions. The contents of this file would look 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@" would be your individual download area identification string found in your Welcome Package.

After that, a run of `apt-get update` is needed. Then, the Kubernetes Plugin can be installed using `apt-get install bacula-enterprise-kubernetes-plugin`

On Redhat/CentOS 7 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
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
```

Then perform a `yum update` and the Kubernetes Plugin package can be installed with `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.

**See also:**

Go to:

- *bacula-backup Proxy Pod Image Installation*
- *Advanced Bacula Backup Proxy Pod Deployment*

Go back to the *Kubernetes Installation* page.

Go back to the main *Kubernetes Plugin page*.

## bacula-backup Proxy Pod Image Installation

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

This image should be installed manually on your local Docker images registry service which is available on your Kubernetes 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 images registry service. The exact procedure depends on your Kubernetes cluster deployment, so make sure to verify the above before attempting to run the Docker commands.

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

Depending on your cluster configuration it may be necessary to set the **baculaimage=<name>** plugin parameter (see section *Backup and Restore Plugin Parameters* 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 will need to modify the Bacula Backup Proxy Pod Image is in the case where your registry requires authentication. See *Advanced Bacula Backup Proxy Pod Deployment* for more details.

**See also:**

Go back to:

- *Kubernetes Plugin Installation with Package Manager*

Go to:

- *Advanced Bacula Backup Proxy Pod Deployment*

Go back to the *Kubernetes Installation* page.

Go back to the main *Kubernetes Plugin page*.

### Advanced Bacula Backup Proxy Pod Deployment

**Warning:** This is an advanced topic related to Kubernetes clusters. It is strongly advised against attempting to modify or configure the Bacula Kubernetes Plugin unless you possess a deep understanding of the process.

You can customize the service parameters used for deploying Bacula backup Pods dedicated to Persistent Volume Claim data backup to suit your needs. The plugin uses the following Pod service deployment YAML template to execute the proxy operation pod on the cluster.

```
apiVersion: v1
kind: Pod
metadata:
  name: {podname}
  namespace: {namespace}
  labels:
    app: {podname}
spec:
  hostname: {podname}
  {nodenameparam}
  containers:
  - name: {podname}
    resources:
      limits:
        cpu: "1"
        memory: "64Mi"
      requests:
        cpu: "100m"
        memory: "16Mi"
    image: {image}
    env:
    - name: PLUGINMODE
      value: "{mode}"
    - name: PLUGINHOST
      value: "{host}"
    - name: PLUGINPORT
      value: "{port}"
    - name: PLUGINTOKEN
      value: "{token}"
    imagePullPolicy: {imagepullpolicy}
    volumeMounts:
```

(continues on next page)

(continued from previous page)

```
    - name: {podname}-storage
      mountPath: /{mode}
restartPolicy: Never
volumes:
  - name: {podname}-storage
    persistentVolumeClaim:
      claimName: {pvcname}
```

The above template uses a number of predefined placeholders which will be replaced by corresponding variables during Pod execution preparation. To customize proxy Pod deployment you can change or tune template variables or the template body. Below is a list of all supported variables with short descriptions and requirement conditions.

**podname** This is the predefined Pod name used by a 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.

**nodenameparam** 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.

**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.

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

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

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

**token** This is an Authorization Token (randomly generated). This variable is required and cannot be customized.

**pvcname** 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 the custom one using the `$DEFAULTPODYAML` environment variable.

**See also:**

Go back to:

- *Kubernetes Plugin Installation with Package Manager*
- *bacula-backup Proxy Pod Image Installation*

Go back to the *Kubernetes Installation* page.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go back to:

- *Features*

Go to:

Go back to the main *Kubernetes Plugin page*.

# 3 Configuration

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

The following Kubernetes 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>** points to the location of the config file which defines how to connect to the Kubernetes cluster. Please check the Kubernetes documentation for more details about the *kubeconfig* file. If this directive is omitted and no other access method is configured then a default config file location will be used - `$HOME/.kube/config`.

This parameter is optional.

**incluster** if this directive is defined then a standard in-cluster access to the Kubernetes API will be used. This option should be used only when the Bacula File Daemon and Kubernetes Plugin are deployed as a Kubernetes cluster service in a Pod. In this case, Kubernetes itself will provide the required access credentials. This option won't work when executed outside of a Kubernetes cluster and services.

This parameter is optional.

**host=<url-k8s-api>** defines a Kubernetes API url. This option is used only with **token** parameter described below. If this option is specified, then both parameters are required.

This parameter is optional.

**token=<bearer-token>** defines a `Bearertoken` used for authorization to the Kubernetes API available at **host=<url-k8s-api>**. This option is used only with **host** parameter described above. You can read more about this type of authentication at: https://swagger.io/docs/specification/authentication/bearer-authentication/

This parameter is optional.

**verify_ssl[=<0 or 1>]** specifies whether or not the plugin should verify a Kubernetes API certificate when the connection uses SSL/TLS. If set to **verify_ssl=0** then verification will be disabled. This is useful when connecting to a Kubernetes API server with a self-signed certificate. The default behavior if this parameter if not set is to perform proper certificate validation.

This parameter is optional.

**ssl_ca_cert=</path/to/file>** specifies a file with the CA certificate used to customize the Kubernetes API server identity to verify the peer.

This parameter is optional.

**timeout=<seconds>** specifies the number of seconds for various network operations. Examples include: waiting for Bacula Backup Proxy Pod connection or Kubernetes API operations, waiting for Pod execution or removal. The default is 600 seconds. The minimum timeout you may set is 1 second. When an invalid value is set the default will be used.

This parameter is optional.

**debug[=1,2,3]** specifies that the plugin backend will generate an execution debug file at location `/bacula/working/backendplugin/`. This file can help with troubleshooting the job execution if something goes wrong. If not defined then no debug file will be generated. From version **16.0.14**, users have the ability to choose from various debug levels, with the highest number encompassing the previous level.

**1:** Save debug messages of k8s server interactions. **2:** Save communication messages with bacula core except data packages. **3:** Save all debug messages.

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

**See also:**

Go to:

- *Estimate and Backup Plugin Parameters*
- *Backup and Restore Plugin Parameters*
- *FileSet Examples*
- *Restore Plugin Parameters*

Go back to the *Kubernetes Configuration page*.

Go back to the main *Kubernetes Plugin page*.

## 3.2 Estimate and Backup Plugin Parameters

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

**namespace=<name>** specifies a Kubernetes namespace name which you want to backup. Multiple `namespace=<name>` parameters are allowed if you want to backup multiple namespaces. If a namespace with `name` can not be found its backup will be silently ignored. If this parameter is not set, all the objects in all namespaces will be saved. If doing a pvcdata backup, the namespace(s) must be specified, and the persistent volume(s) in the specified namespace(s) are included in the backup.

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

**persistentvolume=<name>** specifies a Kubernetes persistent volume configuration you want to backup. Multiple `persistentvolume=<name>` parameters are allowed if you want to backup multiple volumes. You can use standard shell wildcard pattern matching to select multiple volumes using a single `persistentvolume` parameter. If a persistent volume with `name` can not be found its backup 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.

This parameter is optional.

**pvconfig=[0|1]** this option is used to disable persistent volume configuration backups when **pvconfig=0** is set. The default is to backup persistent volume configurations.

This parameter is optional.

**storageclass=<name>** specifies a Kubernetes Storage Class configuration to be backed up. Multiple `storageclass=<name>` parameters are allowed if you want to backup multiple objects. You can use standard shell wildcard pattern matching to select multiple volumes using a single `storageclass` parameter. If a storage class with `name` can not be found, its backup 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]** this option is used to disable Storage Class configuration backups when **scconfig=0** is set. The default is to backup Storage Class object configurations.

This parameter is optional.

**pvcdata[=<pvcname1>,<pvcname2>]** specifies a Kubernetes Persistent Volume Claim names you want to make PVC Data archive for. As all PVCs are namespaced objects, to use this option you should specify the required namespace(s) with **namespace=<name>** parameter. If you define a simple **pvcdata** parameter without the equals sign ("=") and subsequent value, all detected persistent volume claims 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.

**backup_mode=[snapshot|clone|standard]** specifies the general backup mode you want to do backup.

**Note:** The **bacula/backup.mode** pod annotation has precendence over this parameter in the defined pvcs in the pod.

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 in this case.

**See also:**

Go back to:

- *Generic Plugin Parameters*

Go to:

- *Backup and Restore Plugin Parameters*
- *FileSet Examples*
- *Restore Plugin Parameters*

Go back to the *Kubernetes Configuration page*.

Go back to the main *Kubernetes Plugin page*.

## 3.3 Backup and Restore Plugin Parameters

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 Kubernetes 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 Kubernetes 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 Kubernetes 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 Kubernetes 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 Kubernetes 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 Kubernetes pod image pull policy during Bacula Backup Proxy Pod Container deployment. You can configure Kubernetes 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.

**backup_mode=<Standard|Clone|Snapshot>** sets the mode to perform backup of the persistent volume data (pvcdata backup). When you specify this parameter, you are limited with compatibility of the storage using this mode. 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: Snapshot, Clone and Standard.

**See also:**

Go back to:

- *Generic Plugin Parameters*
- *Estimate and Backup Plugin Parameters*

Go to:

- *FileSet Examples*
- *Restore Plugin Parameters*

Go back to the *Kubernetes Configuration page*.

Go back to the main *Kubernetes Plugin page*.

## 3.4 FileSet Examples

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

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

In this example, we will backup a single Kubernetes 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 pvcdata 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 pvcdata=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 Kubernetes cluster:

```
FileSet {
  Name = FS_Kubernetes_incluster
  Include {
    Plugin = "kubernetes: incluster namespace=test pvcdata \
        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 Kubernetes cluster.

**See also:**

Go back to:

- *Generic Plugin Parameters*
- *Estimate and Backup Plugin Parameters*
- *Backup and Restore Plugin Parameters*

Go to:

- *Restore Plugin Parameters*

Go back to the *Kubernetes Configuration page*.

Go back to the main *Kubernetes Plugin page*.

## 3.5 Restore Plugin Parameters

During restore, the Kubernetes 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 *Generic Plugin Parameters* and *Backup and Restore Plugin Parameters*. 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 or YAML>** specifies the file format when restoring to a local filesystem as described above.

This parameter is optional.

**See also:**

Go back to:

- *Generic Plugin Parameters*
- *Estimate and Backup Plugin Parameters*
- *Backup and Restore Plugin Parameters*
- *FileSet Examples*

Go back to the *Kubernetes Configuration page*.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go back to:

- *Features*
- *Installation*

Go to:

- *Operations*
- *Limitations*
- *Troubleshooting*

Go back to the main *Kubernetes Plugin page*.

# 4 Operations

The following article aims at describing Kubernetes Plugin operations.

## 4.1 Backup

The plugin can back up a number of Kubernetes 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 Kubernetes Object which is saved. For PVC Data backup functionality the Kubernetes Plugin generates a data archive as a single `<pvcname>.tar` archive file. The objects are organized inside the Bacula catalog to facilitate browsing and restore operations. In the Bacula catalog, Kubernetes objects are represented as follows:

- `/@kubernetes/namespaces/<namespace>.yaml` - Namespace definition
- `/@kubernetes/namespaces/<namespace>/<objecttype>/<name>.yaml` - Object definitions in the namespace
- `/@kubernetes/namespaces/<namespace>/persistentvolumeclaim/<pvcname>.tar` - PVC Data backup in the selected namespace
- `/@kubernetes/persistentvolumes/<pvname>.yaml` - Persistent Volume definition
- `/@kubernetes/storageclass/<scname>.yaml` - Storage Class definition

All supported Kubernetes Objects will be saved if no filter options are set. You may limit which objects are saved using filtering options described in the *Estimate and Backup Plugin Parameters* chapter. By default, if no filter options are set, all supported Kubernetes Objects will be saved. To see the Kubernetes Objects that may be filtered, a listing mode is available. This mode is described in the *Listing* chapter.

Read more:

### Persistent Volume Claim Backup

All Pods in Kubernetes 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 Kubernetes 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 Kubernetes Plugin which manages secure access to Persistent Volume data for the plugin. It is executed on the Kubernetes cluster infrastructure and requires a network connection to the Kubernetes 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 Kubernetes backup solution (Bacula Enterprise service with Kubernetes Plugin) directly from your on-premise backup infrastructure to backup a public Kubernetes cluster (it requires a simple port forwarding firewall rule) or use public backup infrastructure to back up on-premise Kubernetes cluster(s). Support for these varied architecture modes is built into the Kubernetes Plugin. It is designed to be a *One-Click* solution for Kubernetes 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 Kubernetes Pod. This removes a previous limitation in this area.

Read more:

## PVC Backup Modes

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 Kubernetes 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 Kubernetes 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.

**See also:**

Go to:

- *PVC Backup Flow*
- *Kubernetes Pod Annotations*

Go back to the *Persistent Volume Claim Backup page*.

Go back to the *Kubernetes Backup page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

## PVC Backup Flow

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

1. **Pod annotations:**
   - If the parameter **pvcdata** is defined in fileset and a pod has **bacula** annotations referencing to pvc backup, these pvcs in the pod annotations will be backed up, even if they are not defined in **pvcdata** parameter.
   - The plugin looks for all the pods in the namespace specified in the fileset, if they have pod annotation, the plugin includes the persistent volumes in pod annotation with their respective backup_mode.
   - If a pvc is defined in a pod annotation, the plugin will use the backup mode specified in the pod annotation.
   - If **pvcdata*** parameter is defined in fileset without any pvc specified, the plugin will perform backup of all pvc inside the defined namespaces.
   - If a persistent volume is included in both pod annotation and the FileSet pvcdata option, the pod annotation takes precedence over the FileSet pvcdata option.
2. **Backup mode:**
   - By default, the backup mode is snapshot.
   - The priority of 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.
   - In case that the pvc is not compatible with snapshot, the plugin will try to perform the backup with clone mode.
   - If the pvc clone does not have any data (the backup result of the pvc clone is empty), the plugin will attempt to perform the backup using the standard mode, as long as the pvc is not compatible with the clone.
   - The best option to get consistent data is the snapshot mode.

From Bacula Enterprise version **16.0.14**, the plugin does not perform a backup of pvc data when they are in `Pending` state.

**See also:**

Go back to:

- *PVC Backup Modes*

Go to:

- *Kubernetes Pod Annotations*

Go back to the *Persistent Volume Claim Backup page*.

Go back to the *Kubernetes Backup page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

## Kubernetes Pod Annotations

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 at the Pod you want to backup, the preferred backup mode for the Pod, and what commands you want to execute.

The Pod Annotation feature allows administrators of a Kubernetes Cluster, or any Kubernetes 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 type of backup follows the flow described *here*.

**bacula/backup.volumes:<pvcname[,pvcname2…]>** defines what volumes will be backed up from this Pod. Multiple PVC names may be selected as a comma separated list. This annotation is required if you want to backup volumes on this Pod. Any other annotations are optional.

**bacula/run.before.job.container.command:<container>/<command>** defines what command to execute on which container of the Pod before the volume cloning (or snapshot) and data backup of this Pod occurs. An asterisk (`*`) as <`container`> name means to execute <`command`> on all containers.

**bacula/run.before.job.failjobonerror:[yes|no]** defines whether or not 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 what command to execute on which container of the Pod after volume cloning(or snapshot) and data backup of this Pod. An asterisk (`*`) as <`container`> name means to execute <`command`> on all containers.

**bacula/run.after.job.failjobonerror:[yes|no]** defines whether or not 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 what command to execute on which container of the Pod after volume cloning (or snapshot) and just before any data backup. An asterisk (`*`) as <`container`> name means to execute <`command`> on all containers. Note that the **snapshot** mode will not perform a volume snapshot but rather a volume clone.

**bacula/run.after.snapshot.failjobonerror:[yes|no]** defines to fail the job when exit code of the executed command is not zero. The default is no.

Pod annotations is an extension to the current PVC Data backup feature available with the **pvcdata=…** plugin parameter as described in *Estimate and Backup Plugin Parameters*. 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.

All you need to use a new feature is to configure selected Pod annotations and make sure that the backup for a required Kubernetes namespace is properly configured. There is no need for any plugin configuration modifications.

> **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.

Read more:

### Pod Annotations Examples

Here are a few examples demonstrating how to set up Bacula annotations within Kubernetes Pods.

When dealing with Kubernetes 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:
      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 you will use a simple Linux command `sync` 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 you will use PostgreSQL's database data files quiesce feature to perform consistent backup with volume cloning.

**Note:** The final PostgreSQL backup solution requires more configuration and preparation which was skipped in this example to make it clearer.

The initial directive (*run.before.job.container.command*) halts any 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
```

(continues on next page)

(continued from previous page)

```
    volumeMounts:
      - name: pgsql
        mountPath: /var/lib/pgsql
    volumes:
      - name: pgsql
        persistentVolumeClaim:
          claimName: pgsql-volume
```

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

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

**See also:**

Go to: *Run Container Command Annotation*.

Go back to the *Kubernetes Pod Annotations page*.

Go back to the *Persistent Volume Claim Backup page*.

Go back to the *Kubernetes Backup page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

### Run Container Command Annotation

All flavors of the Run Container Command parameters are remotely executed using the Kubernetes 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.

**See also:**

Go to: *Pod Annotations Examples*.

Go back to the *Kubernetes Pod Annotations page*.

Go back to the *Persistent Volume Claim Backup page*.

Go back to the *Kubernetes Backup page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go back to:

- *PVC Backup Modes*
- *PVC Backup Flow*

Go back to the *Persistent Volume Claim Backup page*.

Go back to the *Kubernetes Backup page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go to:

- *CSI Volume Features Support*

Go back to the *Kubernetes Backup page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

### CSI Volume Features Support

Starting from Bacula Enterprise version **12.6.0**, the support for Kubernetes CSI Volume Cloning functionality was introduced, along with various other features. Subsequently, from Bacula Enterprise version **16.0.8** Kubernetes CSI Volume Snapshot functionality was included. The plugin now automatically detects its ability to perform this functionality starting from this version. As of version **16.0.13**, users have the option to select the backup mode that suits their needs while ensuring compatibility.

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

Additionally, you can configure remote command execution on 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 have implemented the volume cloning or snapshot functionality. The reference to clone feature is CSI Volume Cloning. The reference to snapshot feature is 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, our plugin uses the volume clone and snapshot api. Given the higher level of consistency associated with snapshotting, our plugin uses this technique by default as long as it is supported by the persistent volume CSI driver.

**See also:**

Go back to:

- *Persistent Volume Claim Backup*

Go back to the *Kubernetes Backup page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go to:

- *Restore*
- *Listing*
- *Query Commands*

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

## 4.2 Restore

The Kubernetes Plugin provides two targets for restore operations:

### Restore to Kubernetes Cluster

To use this restore method, the **where=/** parameter of a Bacula `restore` command is used. You can select any supported Kubernetes Object to restore, or batch restore the whole namespace or even multiple namespaces. If you select a single object to restore it will be restored as is without any dependent objects. In most cases, for (`Config Maps`, `Secrets`, `Services`, etc.) this is fine and restore will always succeed. When you restore compound objects (`Pods`, `Deployments`, `Ingress`, etc.), they won't be ready unless all dependencies are resolved. In this case you should make sure that you select all required objects to restore. 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 will generate new ones. From Bacula Enterprise version **16.0.14**, there is no need for concern as the plugin will handle the restore seamlessly.

In Kubernetes, a successful object restore doesn't necessarily result in the service successfully coming online. In some cases further monitoring and investigation will 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 example cases above must be resolved by the Kubernetes administrator. When all issues are resolved, the object should automatically come online. If not, it may be necessary to repeat a restore to redeploy the Object configuration.

The Kubernetes Plugin does not wait for a Object to come online during restore. It checks the Object creation or replace operation status and reports any errors in the job log. The only exception to this is PVC Data restore, when the Kubernetes Plugin will wait for a successful archive data restore. From Bacula Enterprise version **16.0.14**, this operation is always executed before create compound components (`Pods`, `Deployments`, etc.).

**See also:**

Go to:

- *Restore to Local Directory*
- *Restore PVC Data*
- *Restore Examples*

Go back to the *Kubernetes Restore page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

### Restore to Local Directory

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 Kubernetes Plugin are installed. If the path does not exist, it will be created by the Kubernetes Plugin. With this restore mode, you can restore any saved Kubernetes Object including PVC Data archive file to a location on disk.

**See also:**

Go back to:

- *Restore to Kubernetes Cluster*

Go to:

- *Restore PVC Data*
- *Restore Examples*

Go back to the *Kubernetes Restore page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

### Restore PVC Data

Here we describe functionalities and requirements related to pvcdata restore.

### Local Directory Restore

The procedure to restore a PVC Data archive file to a local directory is basically the same as restoring the Kubernetes Object configuration file as described in *Restore to Local Directory Example*. However, output transformation is unavailable and ignored when restoring PVC data. Restore of this data will create a `tar` archive file you can manually inspect and use.

### 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 setup it again. If your endpoint parameters have changed you can update them using Bacula plugin restore options modification as in 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 pvcdata 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
config:             *None*              (*None*)
```

(continues on next page)

(continued from previous page)

```
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
```

You can restore all data available from the backup archive for a selected Persistent Volume Claim and all data will be overwritten, ignoring the `Replace` job parameter. Please take note of this behavior, which may change in the future.

**See also:**

Go back to:

- *Restore to Kubernetes Cluster*
- *Restore to Local Directory*

Go to:

- *Restore Examples*

Go back to the *Kubernetes Restore page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

## Restore Examples

The following articles aims at presenting the examples of restore configuration.

### Restore to Kubernetes Cluster Example

To restore Kubernetes objects to a Kubernetes 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 where=/
...
$ cd /@kubernetes/namespaces/plugintest/configmaps/
cwd is: /@kubernetes/namespaces/plugintest/configmaps/
$ ls
plugintest-configmap.yaml
$ add *
1 file marked.
$ done
Bootstrap records written to /opt/bacula/working/bacula-devel-dir.restore.1.bsr
```

(continues on next page)

(continued from previous page)

```
The Job will require the following (*=>InChanger):
Volume(s)                 Storage(s)                SD Device(s)
===========================================================================

Vol005                    File1                     FileChgr1

Volumes marked with "*" are in the Autochanger.


1 file selected to be restored.

Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/bacula-devel-dir.restore.1.bsr
Where:           /
Replace:         Always
FileSet:         Full Set
Backup Client:   bacula-devel-fd
Restore Client:  bacula-devel-fd
Storage:         File1
When:            2019-09-30 12:39:13
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
Plugin Restore Options
config:             radekk/.kube/config  (*None*)
host:               *None*               (*None*)
token:              *None*               (*None*)
username:           *None*               (*None*)
password:           *None*               (*None*)
verify_ssl:         *None*               (True)
ssl_ca_cert:        *None*               (*None*)
outputformat:       *None*               (RAW)
Use above plugin configuration? (yes/mod/no): mod
```

(continues on next page)

(continued from previous page)

```
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 (1-8): 1
Please enter a value for config: /root/.kube/config
Plugin Restore Options
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)
Use above plugin configuration? (yes/mod/no): yes
Job queued. JobId=1084
```

The plugin does not wait for Kubernetes Objects to become ready and online in the same way as the `kubectl` or the `oc` commands.

**See also:**

Go to: *Restore to Local Directory Example*.

Go back to *Kubernetes Restore Examples*.

Go back to the *Kubernetes Restore page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

## Restore to Local Directory Example

It is possible to restore any Kubernetes 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/
```

(continues on next page)

(continued from previous page)

```
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)
===========================================================================

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*)
```

(continues on next page)

(continued from previous page)

```
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 (1-8): 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:          JSON                (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/plugintest/plugintest.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,
        "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`.

**See also:**

Go to: *Restore to Kubernetes Cluster Example*.

Go back to *Kubernetes Restore Examples*.

Go back to the *Kubernetes Restore page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go back to:

- *Restore to Kubernetes Cluster*

- *Restore to Local Directory*
- *Restore PVC Data*

Go back to the *Kubernetes Restore page*.

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go back to:

- *Backup*

Go to:

- *Listing*
- *Query Commands*

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

## 4.3 Listing

The Bacula Enterprise Kubernetes 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 Kubernetes Objects such as:

- List of Kubernetes Namespaces and their resources.
- List of Kubernetes Persistent Volumes.
- List of Kubernetes Storage Classes Objects.
- List of Kubernetes Cluster Roles Objects.
- List of Kubernetes Cluster Roles Bindings Objects.
- List of Kubernetes VOlume Snapshot Classes Objects.

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 Kubernetes plugin installed.

**plugin=<plugin>** A Plugin name, which would be **kubernetes:** 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 resources types available to list.

**/namespaces** to display a list of Namespaces.

**/namespaces/<namespace>/** to display resources types available to list.

**/namespaces/<namespace>/<resources>/** to display Kubernetes Objects of this Kubernetes Resource.

**/persistentvolumes** to display a list of Persistent Volumes

**/storageclasses** to display a list of Storage Class Objects.

**/clusterroles** to display a list of Cluster Roles Objects.

**/clusterrolebindings** to display a list of Cluster Role Bindings Objects.

**/volumesnapshotclasses** to display a list of Volume Snapshot Classes Objects.

To display available Kubernetes Resources, follow the following command example:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/
Connecting to Client kubernetes-fd at localhost:9102
drwxr-xr-x   1 root     root                    0 2024-10-18 13:56:06  /clusterroles/
drwxr-xr-x   1 root     root                    0 2024-10-18 13:56:06  /
↪clusterrolebindings/
drwxr-xr-x   1 root     root                    0 2024-10-18 13:56:06  /namespaces/
drwxr-xr-x   1 root     root                    0 2024-10-18 13:56:06  /
↪persistentvolumes/
drwxr-xr-x   1 root     root                    0 2024-10-18 13:56:06  /storageclasses/
drwxr-xr-x   1 root     root                    0 2024-10-18 13:56:06  /
↪volumesnapshotclasses/
2000 OK estimate files=6 bytes=0
```

To display the list of all available Kubernetes Namespaces, the following command example can be used:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/namespaces
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪cattle-provisioning-capi-system
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪cattle-system
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪cattle-ui-plugin-system
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪cert-manager
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪cluster-fleet-local-local-1a3d67d0a899
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪default
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪fleet-default
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪fleet-local
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪harbor-system
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪kube-node-lease
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪kube-public
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪kube-system
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪local
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪local-path-storage
-rw-r-----   1 root     root                 1024 2024-10-18 13:59:45  /namespaces/
↪longhorn-system
[...]
2000 OK estimate files=36 bytes=36,864
```

To display available Kubernetes Resources belong to a namespace, follow the following command example:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/namespaces/default/
Connecting to Client kubernetes-fd at localhost:9102
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/configmaps/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/daemonsets/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/deployments/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/endpoints/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/ingresses/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/limitranges/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/persistentvolumeclaims/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/pods/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/podtemplates/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/replicasets/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/replicationcontrollers/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/resourcequotas/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/rolebindings/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/roles/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/secrets/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/services/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/serviceaccounts/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/statefulsets/
drwxr-xr-x   1 root     root                      0 2024-10-18 22:53:11  /namespaces/
↪default/volumesnapshots/
2000 OK estimate files=19 bytes=0
```

To display the list of available Namespaced Kubernetes Objects, use the next command(or similar, it depends on Namespaced Kubernetes Resource indicated):

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/namespaces/default/pods/
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root                   1024 2024-10-18 22:56:08  /namespaces/
↪default/pods/web-server-nginx
2000 OK estimate files=1 bytes=1,024
```

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=bacula-devel-fd plugin="kubernetes:" path=/namespaces/mysql/
↪persitentvolumeclaims/
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root             2147483648 2024-10-18 23:12:09  /namespaces/
↪mysql/persistentvolumeclaims/web-server-mysql
2000 OK estimate files=1 bytes=2,147,483,648
```

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

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

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/persistentvolumes/
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root                   1024 2024-10-18 23:15:14  /
↪persistentvolumes/pvc-cd6d3b67-f72b-4451-ae6e-6e9eb5fc9d2e
-rw-r-----   1 root     root                   1024 2024-10-18 23:15:14  /
↪persistentvolumes/pvc-d5fe7d2b-31b2-469d-962f-89ec87fb5c90
-rw-r-----   1 root     root                   1024 2024-10-18 23:15:14  /
↪persistentvolumes/pvc-e712b02c-472e-4fa1-beae-a7dc0db7afbf
-rw-r-----   1 root     root                   1024 2024-10-18 23:15:14  /
↪persistentvolumes/pvc-e8ad52b4-0d9b-4efd-b98f-5334eae34302
-rw-r-----   1 root     root                   1024 2024-10-18 23:15:14  /
↪persistentvolumes/pvc-ed57261f-096e-4b88-b7cb-7d02e896c602
-rw-r-----   1 root     root                   1024 2024-10-18 23:15:14  /
↪persistentvolumes/pvc-f4f9e72c-7e00-4dd3-bc84-9bedc99097d7
-rw-r-----   1 root     root                   1024 2024-10-18 23:15:14  /
↪persistentvolumes/pvc-fa10cbde-56c2-4268-b5ce-4c2afdc7a868
[...]
2000 OK estimate files=32 bytes=32,768
```

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

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/storageclasses/
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root                   1024 2024-10-18 23:21:50  /storageclasses/
↪ceph-block
-rw-r-----   1 root     root                   1024 2024-10-18 23:21:50  /storageclasses/
↪ceph-bucket
-rw-r-----   1 root     root                   1024 2024-10-18 23:21:50  /storageclasses/
↪ceph-filesystem
-rw-r-----   1 root     root                   1024 2024-10-18 23:21:50  /storageclasses/
↪local-path
-rw-r-----   1 root     root                   1024 2024-10-18 23:21:50  /storageclasses/
↪local-storage
-rw-r-----   1 root     root                   1024 2024-10-18 23:21:50  /storageclasses/
↪longhorn
-rw-r-----   1 root     root                   1024 2024-10-18 23:21:50  /storageclasses/
↪nfs-client
2000 OK estimate files=7 bytes=7,168
```

To display the list of all defined Cluster Roles Objects, the following command example can be used:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/clusterroles/
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root                    1024 2024-10-18 23:22:57  /clusterroles/
↪admin
-rw-r-----   1 root     root                    1024 2024-10-18 23:22:57  /clusterroles/
↪calico-node
-rw-r-----   1 root     root                    1024 2024-10-18 23:22:57  /clusterroles/
↪capi-aggregated-manager-role
-rw-r-----   1 root     root                    1024 2024-10-18 23:22:57  /clusterroles/
↪capi-manager-role
-rw-r-----   1 root     root                    1024 2024-10-18 23:22:57  /clusterroles/
↪cattle-fleet-local-system-fleet-agent-role
-rw-r-----   1 root     root                    1024 2024-10-18 23:22:57  /clusterroles/
↪cattle-globalrole-admin
[...]
2000 OK estimate files=164 bytes=167,936
```

To display the list of all defined Cluster Role Bindings Objects, the following command example can be used:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/clusterrolesbindings/
Connecting to Client kubernetes-fd at localhost:9102
-rw-r-----   1 root     root                    1024 2024-10-18 23:23:06  /
↪clusterrolebindings/canal-calico
-rw-r-----   1 root     root                    1024 2024-10-18 23:23:06  /
↪clusterrolebindings/canal-flannel
-rw-r-----   1 root     root                    1024 2024-10-18 23:23:06  /
↪clusterrolebindings/capi-manager-rolebinding
-rw-r-----   1 root     root                    1024 2024-10-18 23:23:06  /
↪clusterrolebindings/cattle-fleet-local-system-fleet-agent-role-binding
-rw-r-----   1 root     root                    1024 2024-10-18 23:23:06  /
↪clusterrolebindings/cattle-globalrolebinding-globalrolebinding-nxl4c
-rw-r-----   1 root     root                    1024 2024-10-18 23:23:06  /
↪clusterrolebindings/cattle-globalrolebinding-grb-5jhgc
[...]
2000 OK estimate files=133 bytes=136,192
```

To display the list of all defined Volume Snapshot Classes Objects, the following command example can be used:

```
*.ls plugin=kubernetes: client=kubernetes-fd path=/volumesnapshotclasses/
Connecting to Client kubernetes-fd at localhost:9102
kubernetes: Listing returned an empty result
2000 OK estimate files=0 bytes=0
```

**See also:**

Go back to:

- *Backup*
- *Restore*

Go to:

- *Query Commands*

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

## 4.4 Query Commands

The Bacula Enterprise Kubernetes Plugin supports the “query command”. This type of command allows the plugin to display useful information about something beyond its main function, such as:

- User permissions on cluster

The feature uses the special **.query** command on bconsole.

This type of commands require the following parameters to be set:

**client=<client>** A Bacula Client name with the Kubernetes plugin installed.

**plugin=”<plugin>”** A Plugin name, which would be **kubernetes:** in this case, with optional plugin parameters as described in section *Generic Plugin Parameters* or *Estimate and Backup Plugin Parameters*. It depends on the function.

**parameter=<function>** The functions will be explained in the following sections.

---

**Note:** We can change the output to **JSON** format. To do that, we write **json** before function and separated them with **|**, like: **json|<function>**.

---

### user_permissions

This function allows us to know if the indicated user in k8s config file has correct permissions to work with this plugin or if this user misses some permission.

In this command, the more important parameters are:

**config=<path>** Path to k8s config file connection.

**namespace=<namespace>** We indicate a namespace if we want to know permissions that the user has inside in that namespace. If you don’t use this parameter, you only query for cluster permissions.

---

**Note:** We can define several **namespace** in the same command

---

Examples:

Listing 1: **Example of the function without namespace**

```
*.query client=kubernetes-fd plugin="kubernetes:" parameter=user_permissions
Allowed to: Cluster list clusterroles -> True
Allowed to: Cluster get clusterroles -> True
Allowed to: Cluster create clusterroles -> True
Allowed to: Cluster update clusterroles -> True
Allowed to: Cluster list clusterrolebindings -> True
Allowed to: Cluster get clusterrolebindings -> True
Allowed to: Cluster create clusterrolebindings -> True
Allowed to: Cluster list namespaces -> True
Allowed to: Cluster get namespaces -> True
Allowed to: Cluster create namespaces -> True
Allowed to: Cluster list storageclasses -> True
```

(continues on next page)

(continued from previous page)

```
Allowed to: Cluster get storageclasses -> True
Allowed to: Cluster create storageclasses -> True
Allowed to: Cluster list volumesnapshotclasses -> True
Allowed to: Cluster get volumesnapshotclasses -> True
Allowed to: Cluster create volumesnapshotclasses -> True
Summary:
Your user has all necessary permissions
```

Listing 2: **Example of the function with some misses permissions**

```
*.query client=kubernetes-fd plugin="kubernetes:" parameter=user_permissions
Allowed to: Cluster list clusterroles -> False
Allowed to: Cluster get clusterroles -> False
Allowed to: Cluster create clusterroles -> False
Allowed to: Cluster update clusterroles -> False
Allowed to: Cluster list clusterrolebindings -> True
Allowed to: Cluster get clusterrolebindings -> True
Allowed to: Cluster create clusterrolebindings -> True
Allowed to: Cluster list namespaces -> True
Allowed to: Cluster get namespaces -> True
Allowed to: Cluster create namespaces -> True
Allowed to: Cluster list storageclasses -> True
Allowed to: Cluster get storageclasses -> True
Allowed to: Cluster create storageclasses -> True
Allowed to: Cluster list volumesnapshotclasses -> True
Allowed to: Cluster get volumesnapshotclasses -> True
Allowed to: Cluster create volumesnapshotclasses -> True
You must grant your user to the next permissions:
- Scope: `Cluster` Resource: `clusterroles` Action: `list`
- Scope: `Cluster` Resource: `clusterroles` Action: `get`
- Scope: `Cluster` Resource: `clusterroles` Action: `create`
- Scope: `Cluster` Resource: `clusterroles` Action: `update`
```

Listing 3: **Example of the function with a namespace**

```
*.query client=kubernetes-fd plugin="kubernetes: debug=1 namespace=\"namespace-1\""␣
↪parameter=user_permissions
Allowed to: Cluster list clusterroles -> True
Allowed to: Cluster get clusterroles -> True
Allowed to: Cluster create clusterroles -> True
Allowed to: Cluster update clusterroles -> True
Allowed to: Cluster list clusterrolebindings -> True
Allowed to: Cluster get clusterrolebindings -> True
Allowed to: Cluster create clusterrolebindings -> True
Allowed to: Cluster list namespaces -> True
Allowed to: Cluster get namespaces -> True
Allowed to: Cluster create namespaces -> True
Allowed to: Cluster list storageclasses -> True
Allowed to: Cluster get storageclasses -> True
Allowed to: Cluster create storageclasses -> True
Allowed to: Cluster list volumesnapshotclasses -> True
Allowed to: Cluster get volumesnapshotclasses -> True
```

(continues on next page)

(continued from previous page)

```
Allowed to: Cluster create volumesnapshotclasses -> True
Allowed to: namespace/namespace-1 list configmaps -> True
Allowed to: namespace/namespace-1 get configmaps -> True
Allowed to: namespace/namespace-1 create configmaps -> True
Allowed to: namespace/namespace-1 update configmaps -> True
Allowed to: namespace/namespace-1 list daemonsets -> True
Allowed to: namespace/namespace-1 get daemonsets -> True
Allowed to: namespace/namespace-1 create daemonsets -> True
Allowed to: namespace/namespace-1 update daemonsets -> True
Allowed to: namespace/namespace-1 list deployments -> True
Allowed to: namespace/namespace-1 get deployments -> True
Allowed to: namespace/namespace-1 create deployments -> True
Allowed to: namespace/namespace-1 update deployments -> True
Allowed to: namespace/namespace-1 list endpoints -> True
Allowed to: namespace/namespace-1 get endpoints -> True
Allowed to: namespace/namespace-1 create endpoints -> True
Allowed to: namespace/namespace-1 list ingresses -> True
Allowed to: namespace/namespace-1 get ingresses -> True
Allowed to: namespace/namespace-1 create ingresses -> True
Allowed to: namespace/namespace-1 list limitranges -> True
Allowed to: namespace/namespace-1 get limitranges -> True
Allowed to: namespace/namespace-1 create limitranges -> True
Allowed to: namespace/namespace-1 list persistentvolumes -> True
Allowed to: namespace/namespace-1 get persistentvolumes -> True
Allowed to: namespace/namespace-1 create persistentvolumes -> True
Allowed to: namespace/namespace-1 list persistentvolumeclaims -> True
Allowed to: namespace/namespace-1 get persistentvolumeclaims -> True
Allowed to: namespace/namespace-1 create persistentvolumeclaims -> True
Allowed to: namespace/namespace-1 list pods -> True
Allowed to: namespace/namespace-1 get pods -> True
Allowed to: namespace/namespace-1 create pods -> True
Allowed to: namespace/namespace-1 delete pods -> True
Allowed to: namespace/namespace-1 list podtemplates -> True
Allowed to: namespace/namespace-1 get podtemplates -> True
Allowed to: namespace/namespace-1 create podtemplates -> True
Allowed to: namespace/namespace-1 list replicasets -> True
Allowed to: namespace/namespace-1 get replicasets -> True
Allowed to: namespace/namespace-1 create replicasets -> True
Allowed to: namespace/namespace-1 update replicasets -> True
Allowed to: namespace/namespace-1 list replicationcontrollers -> True
Allowed to: namespace/namespace-1 get replicationcontrollers -> True
Allowed to: namespace/namespace-1 create replicationcontrollers -> True
Allowed to: namespace/namespace-1 list resourcequotas -> True
Allowed to: namespace/namespace-1 get resourcequotas -> True
Allowed to: namespace/namespace-1 create resourcequotas -> True
Allowed to: namespace/namespace-1 update resourcequotas -> True
Allowed to: namespace/namespace-1 list roles -> True
Allowed to: namespace/namespace-1 get roles -> True
Allowed to: namespace/namespace-1 create roles -> True
Allowed to: namespace/namespace-1 list rolebindings -> True
Allowed to: namespace/namespace-1 get rolebindings -> True
Allowed to: namespace/namespace-1 create rolebindings -> True
```

(continues on next page)

(continued from previous page)

```
Allowed to: namespace/namespace-1 list secrets -> True
Allowed to: namespace/namespace-1 get secrets -> True
Allowed to: namespace/namespace-1 create secrets -> True
Allowed to: namespace/namespace-1 list services -> True
Allowed to: namespace/namespace-1 get services -> True
Allowed to: namespace/namespace-1 create services -> True
Allowed to: namespace/namespace-1 list serviceaccounts -> True
Allowed to: namespace/namespace-1 get serviceaccounts -> True
Allowed to: namespace/namespace-1 create serviceaccounts -> True
Allowed to: namespace/namespace-1 list statefulsets -> True
Allowed to: namespace/namespace-1 get statefulsets -> True
Allowed to: namespace/namespace-1 create statefulsets -> True
Allowed to: namespace/namespace-1 list volumesnapshots -> True
Allowed to: namespace/namespace-1 get volumesnapshots -> True
Allowed to: namespace/namespace-1 create volumesnapshots -> True
Allowed to: namespace/namespace-1 delete volumesnapshots -> True
Summary:
Your user has all necessary permissions
```

Listing 4: **Same command that before but with json output format**

```
*.query client=kubernetes-fd plugin="kubernetes: namespace=\"namespace-1\""␣
↪parameter=json|user_permissions
{"cluster": {"clusterrolebindings":
{"create": true, "get": true, "list": true},
"clusterroles": {"create": true,
"get": true, "list": true, "update": true},
"namespaces": {"create": true, "get": true, "list": true},
"storageclasses": {"create": true, "get": true, "list": true},
"volumesnapshotclasses": {"create": true, "get": true, "list": true}},
"namespaced": {"namespace-1": {
"configmaps": {"create": true, "get": true, "list": true, "update": true},
"daemonsets": {"create": true, "get": true, "list": true, "update": true},
"deployments": {"create": true, "get": true, "list": true, "update": true},
"endpoints": {"create": true, "get": true, "list": true},
"ingresses": {"create": true, "get": true, "list": true},
"limitranges": {"create": true, "get": true, "list": true},
"persistentvolumeclaims": {"create": true, "get": true, "list": true},
"persistentvolumes": {"create": true, "get": true, "list": true},
"pods": {"create": true, "delete": true, "get": true, "list": true},
"podtemplates": {"create": true, "get": true, "list": true},
"replicasets": {"create": true, "get": true, "list": true, "update": true},
"replicationcontrollers": {"create": true, "get": true, "list": true},
"resourcequotas": {"create": true, "get": true, "list": true, "update": true},
"rolebindings": {"create": true, "get": true, "list": true},
"roles": {"create": true, "get": true, "list": true},
"secrets": {"create": true, "get": true, "list": true},
"serviceaccounts": {"create": true, "get": true, "list": true},
"services": {"create": true, "get": true, "list": true},
"statefulsets": {"create": true, "get": true, "list": true},
"volumesnapshots": {"create": true, "delete": true, "get": true, "list": true}
}}}
```

**See also:**

Go back to:

- *Backup*
- *Restore*
- *Listing*

Go back to the *Kubernetes Operations page*.

Go back to the main *Kubernetes Plugin page*.

**See also:**

Go back to:

- *Features*
- *Installation*
- *Configuration*

Go to:

- *Limitations*
- *Troubleshooting*

Go back to the main *Kubernetes Plugin page*.

## 5 Limitations

- Only full level backups are possible. This is a Kubernetes limitation.
- You can perform a single PVC Data backup or restore with a single Bacula File Daemon installation associated with single **fdaddress=<name>**. This limitation may be removed in a future release of the Kubernetes Plugin.
- 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.

**See also:**

Go back to:

- *Features*
- *Installation*
- *Configuration*
- *Operations*

Go to:

- *Troubleshooting*

Go back to the main *Kubernetes Plugin page*.

# 6 Troubleshooting

In this section we describe common problems and ways to solve them.

## 6.1 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 Kubernetes plugin not in a Pod, or Kubernetes does not provide default service access in your installation. In the latter case you should use a standard Kubernetes access method in a prepared kubeconfig file.

**See also:**

Go back to:

- *Features*
- *Installation*
- *Configuration*
- *Operations*
- *Limitations*

Go back to the main *Kubernetes Plugin page*.