Simplifying Backup, Recovery, and Continuous Business Resilience in Kubernetes

Implementation guide for data protection in Kubernetes with Velero, Garage, and Lightbits

August 2025

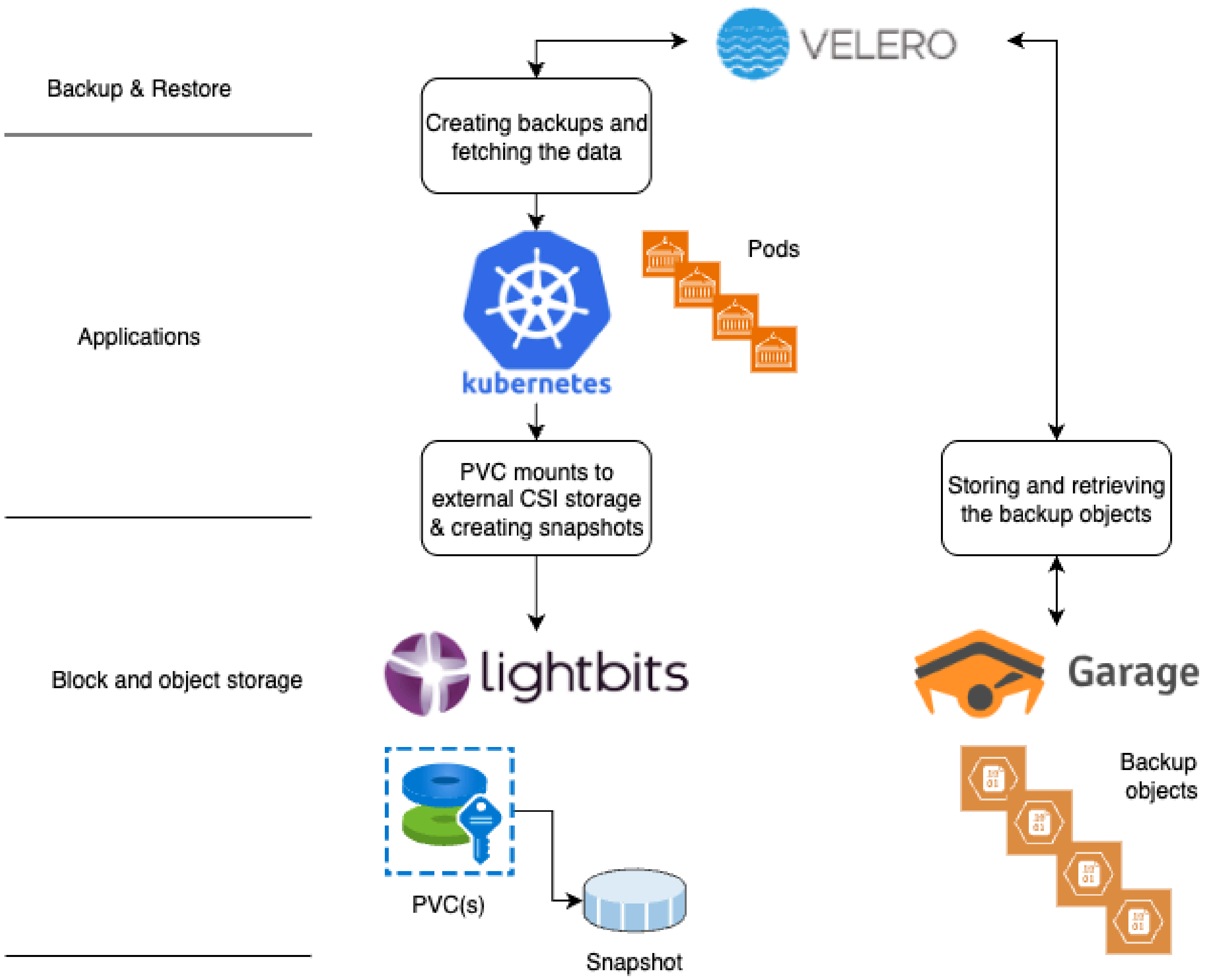
Abstract
This detailed abstract goes beyond simple installation, guiding you through the essential steps of connecting Velero and Garage to work flawlessly with your existing Kubernetes and Lightbits infrastructure. We will demonstrate how to execute backups and restores with confidence, ensuring that all components work together seamlessly. By following this approach, you can protect your critical application data, simplify disaster recovery, and maintain business continuity - all while leveraging the performance and efficiency of Lightbits. Get ready to transform your data protection strategy from a complex challenge into a streamlined, automated process.

Table of Contents

1. Introduction

In this white paper, we will show you how to have a fully integrated backup and restore solution for Kubernetes, running its Physical Volume Claims (PVC) on top of Lightbits' software-defined storage. We took Velero as the backup and restore engine, and Garage as the AWS-compliant S3 object solution to store backups and retrieve restores. The paper is focused on the Velero and Garage installs and how to make them work together.

The diagram below illustrates the implementation architecture.



To make the configuration easy, all the nodes were running Alma Linux 9.6.

2. Installing and Configuring the Garage Object Store

To run Garage as an object store, we will need to have at least three nodes. On these nodes, Docker must be installed and SELinux must be permanently disabled. To disable SELinux temporarily and permanently, use the following commands:

```Shell
Temporarily:
sudo setenforce 0

Permanently:
sudo vi /etc/selinux/config
look for SELINUX=permissive and change to
SELINUX=disabled

Save the file by <Esc>:wq!
```

2.1. Installing Docker

```Shell
sudo dnf install -y docker
```

2.2 Garage Container

The next thing to do is to pull the Garage container:

```Shell
sudo docker pull dxflrs/garage:v2.0.0
```

Garage works with an rpc secret between the nodes. The rpc secret should be created as follows:

```Shell
openssl rand -hex 32

Output: 33f14e516d4f9f48f8bdb210c1e6145cbacefdd08f33045a6d97fb66eb3a35a2
```

2.3 Garage Configuration

Garage should be configured before it can be installed. The /etc/garage.toml file should be created and has the following contents:

```Shell
metadata_dir = "/var/lib/garage/meta"
data_dir = "/var/lib/garage/data"
db_engine = "lmdb"
metadata_auto_snapshot_interval = "6h"

replication_factor = 3

compression_level = 2

rpc_bind_addr = "garage1:3901"
rpc_public_addr = "garage1:3901"
rpc_secret = "33f14e516d4f9f48f8bdb210c1e6145cbacefdd08f33045a6d97fb66eb3a35a2"

[s3_api]
s3_region = "garage"
api_bind_addr = "garage1:3900"
root_domain = ".s3.garage"

[s3_web]
bind_addr = "garage1:3902"
root_domain = ".web.garage"
index = "index.html"
```

Make sure that the rpc is copied as it was created in the configuration file.

Perform the steps above for the second and the first node. Make sure that the server name is changed, in the above example, from garage1 to garage2 and to garage3.

Now that all the nodes are configured, we can start each node by itself:

```Shell
docker run   -d \
--name garaged \
--restart always \
--network host \
-v /etc/garage.toml:/etc/garage.toml \
-v /var/lib/garage/meta:/var/lib/garage/meta \
-v /var/lib/garage/data:/var/lib/garage/data \
dxflrs/garage:v2.0.0
```

We can now create an alias and add it to the .bashrc file.

```Shell
vi ~/.bashrc
add the following line at the bottom
alias garage="docker exec -ti garaged /garage"

And run directly after saving the file:
source ~/.bashrc
```

Next, we can execute the following command to see the status of each Garage node:

```Shell
garage status
```

Before we add the nodes to the cluster, the firewalld must be updated with a new port. Run the following:

```Shell
sudo firewall-cmd --zone=public --add-port=3900/tcp --permanent
sudo firewall-cmd --zone=public --add-port=3901/tcp --permanent
sudo firewall-cmd --zone=public --add-port=3902/tcp --permanent
sudo firewall-cmd --reload
```

2.4 Configuring the Garage Cluster

Each Garage node can now be added to the Garage cluster; each node has its own node ID. In the example below, I used garage1, to add node garage2 and add node garage3.

Follow the steps below:

```
Shell
Go to node garage2 and provide the command

garage node id

output:
e673e5139daca633e1f1e6387e0704d546bc38f652b571de7f118023f8ff505d@garage2:3901

Go to node garage1 and provide the command
garage node connect
e673e5139daca633e1f1e6387e0704d546bc38f652b571de7f118023f8ff505d@garage2:3901

Go to node garage3 and provide the command

garage node id
cec691cc5dc56145519fb87dd0f29aec03ec8c1574e88f6c8d982ebcf898245a@garage3:3901

output:
garage node connect
cec691cc5dc56145519fb87dd0f29aec03ec8c1574e88f6c8d982ebcf898245a@garage3:3901
```

To check that all the nodes are added to the cluster, provide the following command on garage1:

```
Shell
garage status
Output:
==== HEALTHY NODES ====
ID                Hostname  Address             Tags      Zone  Capacity  DataAvail       Version
127ee075ef25f94e  garage1   192.168.1.216:3901
cec691cc5dc56145  garage3   192.168.1.218:3901
e673e5139daca633  garage2   192.168.1.217:3901
```

Garage needs a layout for the Tags, Zone, and Capacity. Provide the following command:

```
Shell
garage layout assign 127e -z gar1 -c 90G -t garage1
garage layout assign e673 -z gar2 -c 90G -t garage2
garage layout assign cec6 -z gar3 -c 90G -t garage3
```

Verify that it is set up correctly:

```Shell
garage layout show
output:
==== CURRENT CLUSTER LAYOUT ====
ID                Tags       Zone  Capacity  Usable capacity
127ee075ef25f94e  [garage1]  gar1  90.0 GB   90.0 GB (100.0%)
cec691cc5dc56145  [garage3]  gar3  90.0 GB   90.0 GB (100.0%)
e673e5139daca633  [garage2]  gar2  90.0 GB   90.0 GB (100.0%)

Zone redundancy: maximum

Current cluster layout version: 1
```

Make the layout permanent:

```Shell
garage layout apply --version 1
```

Create the storage bucket in Garage:

```Shell
garage bucket create backup
```

To verify:

```Shell
garage bucket list

Output:
ID                Created     Global aliases  Local aliases
5b58e473d4140ec4  2025-08-20  backup
```

2.5 Configuring the Credentials for the Backup Bucket

The credentials for the backup bucket should be created with the following command:

```Shell
garage key create backup

Output:
==== ACCESS KEY INFORMATION ====
Key ID:              GK89bf439c5971daabb685ea75
Key name:            backup
Secret key:
32c1dab9605c18640781c1af91f7720d9684bb417a0db1aa7de9fa46167562b6
Created:             2025-08-20 07:44:17.131 +00:00
Validity:            valid
Expiration:          never
Can create buckets:  false
```

To verify:

```Shell
garage key list

Output:
ID                           Created     Name    Expiration
GK89bf439c5971daabb685ea75  2025-08-20  backup  never
```

Fetch the access key for the backup bucket:

```Shell
garage key info backup

Output:
==== ACCESS KEY INFORMATION ====
Key ID:              GK89bf439c5971daabb685ea75
Key name:            backup
Secret key:          (redacted)
Created:             2025-08-20 07:44:17.131 +00:00
Validity:            valid
Expiration:          never

Can create buckets:  false

==== BUCKETS FOR THIS KEY ====
```

```
Permissions  ID                Global aliases  Local aliases
RWO          5b58e473d4140ec4  backup
```

With the above keys, we can configure the bucket backup for access:

```shell
garage bucket allow --read --write --owner backup --key backup
```

To verify:

```shell
garage bucket info backup

Output:
==== BUCKET INFORMATION ====
Bucket:
5b58e473d4140ec4b35962623c899e2fecbca40c015cc442e6181e2f347678ef
Created:          2025-08-20 07:43:17.653 +00:00

Size:             0 kiB (0 KB)
Objects:          0

Website access:   false

Global alias:     backup

==== KEYS FOR THIS BUCKET ====
Permissions  Access key                          Local aliases
RWO          GK89bf439c5971daabb685ea75  backup
```

2.6 Installing awscli

For Velero to communicate with the Garage object store, you will need to install the awscli. Garage is compliant with the AWS S3 command set, and this is the easiest way to communicate from Velero to Garage:

```Shell
sudo dnf install -y pip
python -m pip install --user awscli
```

After the installation, create the configuration file for awscli:

```Shell
vi ~/.awsrc

export AWS_ACCESS_KEY_ID=GK89bf439c5971daabb685ea75
export AWS_SECRET_ACCESS_KEY=32c1dab9605c18640781c1af91f7720d9684bb417a0db1aa7de9fa46167562b6
export AWS_DEFAULT_REGION='garage'
export AWS_ENDPOINT_URL='http://garage1:3900'
```

Copy the ~/.awsrc to garage2 and garage3, and change the AWS_ENDPOINT_URL to garage2 and garage3.

To be able to connect to the Garage storage, enter the following command:

```Shell
source ~/.awsrc
```

This will allow you to connect with the Garage object store with the AWS S3 command set. To verify, enter the following:

```Shell
aws s3 ls

Output:
2025-08-20 09:43:17 backup
```

3. Installing and Configuring Velero

In this setup, we have used the Velero from Github. To download the Velero software, download the tar file from this location: https://github.com/vmware-tanzu/velero/releases
In this case, we have used the tar file: `velero-v1.16.2-linux-amd64.tar.gz`.

The next thing to do is to extract this file:

```shell
tar -xvf velero-v1.16.2-linux-amd64.tar.gz

This will create the directory velero-v1.16.2-linux-amd64

Please go to that directory, all the configurations will be stored here.
```

Create the `credentials-velero` file - to access the Garage object store - with the following contents:

```shell
vi ./credentials-velero

aws_access_key_id = GK89bf439c5971daabb685ea75
aws_secret_access_key =
32c1dab9605c18640781c1af91f7720d9684bb417a0db1aa7de9fa46167562b6
aws_default_region = 'garage'
aws_endpoint_url = 'http://garage1:3900'
```

The next step is to install the kubectl CLI to connect to the already existing Kubernetes cluster:

```shell
sudo dnf install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

Then, configure the `~.kube/config` file with the correct certificates to make sure that Velero can connect to the Kubernetes cluster.

For example:

Shell

```
vi ~.kube/config

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJkakNDQVIyZ0F3SUJBZ0lCQURBS0JnZ3Foa2p
PUFFRREFqQWpNU0V3SHdZRFZRUUREQmhyTTNNdGMyVnkKZG1WeUxXTmhRREUzTkRnNE56azFPRE13SG
hjTk1qVXdOakF5TVRVMU16QXpXaGNOTXpVd05UTXhNVFUxTXpBegpXakFqTVNFd0h3WURWUVFEREJoc
k0zTXRjMlZ5ZG1WeUxXTmhRREUzTkRnNE56azFPRE13V1RBVEJnY3Foa2pPC0lBRSUJCZ2dxaGtqT1BR
TUJCd05DQUFUaHp3aWVudWJiV0krZThyMFRQMk5GKzdkUGtFUjRnZFRkbGZYcWNIYVgKY21ldEp6aFo
3dlZweE5SU1BCV1g4dCtjMmJ0NC9SaGd2VktRYzFZWUo0SmxvMEl3UURBT0JnTlZIUThCQWY4RQpCQU
1DQXFRd0R3WURWUjBUQVFIL0JBVXdBd0VCL3pBZEJnTlZIUTRFRmdRVTJyVFJDbG9Ta1JUZU5yUmNDR
FFPCm4xZmhkS1l3Q2dZSUtvWkl6ajBFQXdJRFJ3QXdSQUlnQTVpL05aUERjcjlZbWI4dlJDK1JRWGtt
aUZ2YzhtV0wKTVg4QVR1TmhET1VDSUNtTm9zR3gvc2JsOWpvVUFGUzZmN2MzQUxLOG0vVGJILzNSYng
zaEV6elUKLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    server: https://192.168.1.32:6443
  name: default
contexts:
- context:
    cluster: default
    user: default
  name: default
current-context: default
kind: Config
preferences: {}
users:
- name: default
  user:
    client-certificate-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJrVENDQVRlZ0F3SUJBZ0lJQVRGU0hrWU5MWEV
3Q2dZSUtvWkl6ajBFQXdJd0l6RWhNQjhHQTFVRUF3d1kKYXpOekxXTnNhV1Z1ZEMxallVQXhOelE0T0
RjNU5UZ3pNQjRYRFRJMU1EWXdNakUxTlRNd00xb1hEVEkyTURZdwpNakUxTlRNd00xb3dNREVYTUJVR
0ExVUVDaE1PYzNsemRHVnRPbTFoYzNSbGNuTXhGVEFUQmdOVkJBTVRESE41CmMzUmxiVHBoWkcxcGJJq
QlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VIQTBJQUJEYkw1Skc2UUhsNWlzZkUKY05Pd2dNd2V
GRGRHMUYvdHV6VDBBUHI0SGljdnFEK1FNMHV4TE43UVdrelQ2YWo5Y2ozUFNxeXhaa21vTHVQcgplOS
tTTDlDalNEQkdNQTRHQTFVZER3RUIvd1FFQXdJRm9EQVRCZ05WSFNVRUREQUtCZ2dyQmdFRkJRY0RBa
kFmCkJnTlZIU01FR0RBV2dCUWt2eVN6OFZPOTRSSTjhhRWZOcGdtczlRcWR6ekFLQmdncWhrak9QUVFE
QWdOSUFEQkYKQWlFQXJFelo4cGF0L2xGeW03a2xCWmw4S1E0b2xoL2EvVXpVZVUrNURYM08rL1FDSUR
xZkpiTWZFWndhZWNzVApmVngweEl5SWg3WnJkMEI3cFI0bHFtakJ2WElhCi0tLS0tRU5EIENFUlRJRk
lDQVRFLS0tLS0KLS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJkekNDQVIyZ0F3SUJBZ0lCQ
URBS0JnZ3Foa2pPUFFRREFqQWpNU0V3SHdZRFZRUUREQmhyTTNNdFkyeHBAKWlc1MExXTmhRREUzTkRn
```

```
NE56azFPRE13SGhjTk1qVXdOakF5TVRVMU16QXpXaGNOTXpVd05UTXhNVFUxTXpBegpXakFqTVNFd0h
3WURWUVFEREJock0zTXRZMnhwWlc1MExXTmhRREUzTkRnNE56azFPRE13V1RBVEJnY3Foa2pPClBRSU
JCZ2dxaGtqT1BRTUJCd05DQUFRNGx4SzliRkdOOE92TUNNbSt5V05TNEFCamNoSnBKclBqZmM1b0M2Q
XMKMW5Yb3EyWU1zRSs1T0cyY3FtbHQwOEpBeXVtekJyejhzd2tjdWgvcnZhM3RvMEl3UURBT0JnTlZI
UThCQWY4RQpCQU1DQXFRd0R3WURWUjBUQVFIL0JBVXdBd0VCL3pBZEJnTlZIUTRFRmdRVUpMOGtzL0Z
UdmVFVGZHaEh6YVlKCnJQVUtuYzh3Q2dZSUtvWkl6ajBFQXdJRFNBQXdSUUloQUlUZm5iT202Q1R5MV
VmR0pUMjJSNGt5aWlLcGE3dzUKbk13NkhZT3YrU284QWlBbFNRckNWTTJ4ZEVMYW9qQzR5TGxvVXVIY
TlOejBSMGlFbmNZeWo2M0x3dz09Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
    client-key-data:
LS0tLS1CRUdJTiBFQyBQUklWQVRFIEtFWS0tLS0tCk1IY0NBUUVFSUFGYzV5RjVyN0dlWi9hcXdUV0R
WMUZtQ1ZYTmNZb0x3NlVCZWtCY2EvbzRvQW9HQ0NxR1NNNDkKQXdFSG9VUURRZ0FFTnN2a2ticEFlWG
1LeDhSdzA3Q0F6QjRVTjBiVVgrMjdOUFFBK3ZnZUp5K29QNUF6UzdFcwozdEJhVE5QcHFQMXlQYzlLc
kxGbVNhZ3U0K3Q3MzVJdjBBPT0KLS0tLS1FTkQgRUMgUFJJVkFURSBLRVktLS0tLQo=
```

Verify that kubectl is working correctly:

```
Shell
kubectl get namespaces

Output:
NAME              STATUS   AGE
acme              Active   84d
default           Active   84d
kube-node-lease   Active   84d
kube-public       Active   84d
kube-system       Active   84d
lightkube         Active   24d
```

Now that we have verified access to Kubernetes and Garage, we can install the Velero containers in Kubernetes. The installation will enable the creation of backups for filesystems and for PVCs with external CSI snapshot providers. There are two plugins provided: the first is for the filesystem and the second is for the external CSI snapshot provider.

The `–use-node-agent` option is for the filesystem. The `–features=EnableCSI` option is for the external CSI snapshot provider.

Use the following command:

Shell

```
velero install \
--provider aws \
--plugins
velero/velero-plugin-for-aws:v1.2.1,velero/velero-plugin-for-csi:v0.4.0 \
--bucket backup \
--secret-file ./credentials-velero \
--use-volume-snapshots=true \
--backup-location-config
region=garage,s3ForcePathStyle="true",s3Url=http://192.168.1.216:3900 \
--use-node-agent \
--features=EnableCSI
```

This will create a new namespace in Kubernetes called velero.

Verify this as follows:

Shell

```
kubeclt get namespaces

Output:
NAME              STATUS   AGE
acme              Active   84d
default           Active   84d
kube-node-lease   Active   84d
kube-public       Active   84d
kube-system       Active   84d
lightkube         Active   24d
velero            Active   4d19h
```

Verify the contents of the namespace velero:

Shell

```
kubectl get pods -n velero

NAME                                          READY   STATUS    RESTARTS        AGE
node-agent-rzm4z                              1/1     Running   3 (125m ago)    4d19h
velero-64b59f5cd5-nbs8r                       1/1     Running   2 (125m ago)    4d1h
```

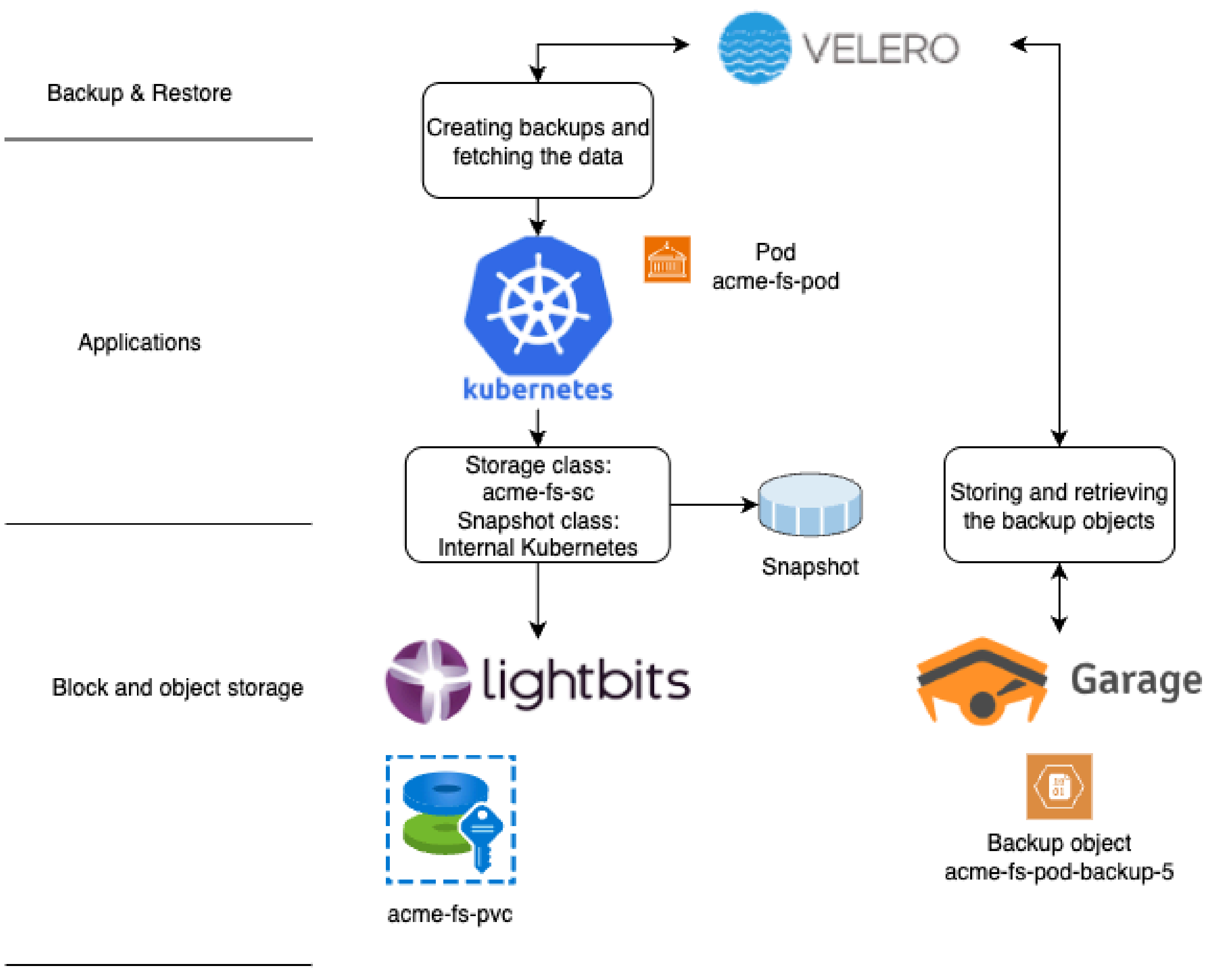
Velero is now ready to create backups and restore.

lightbits

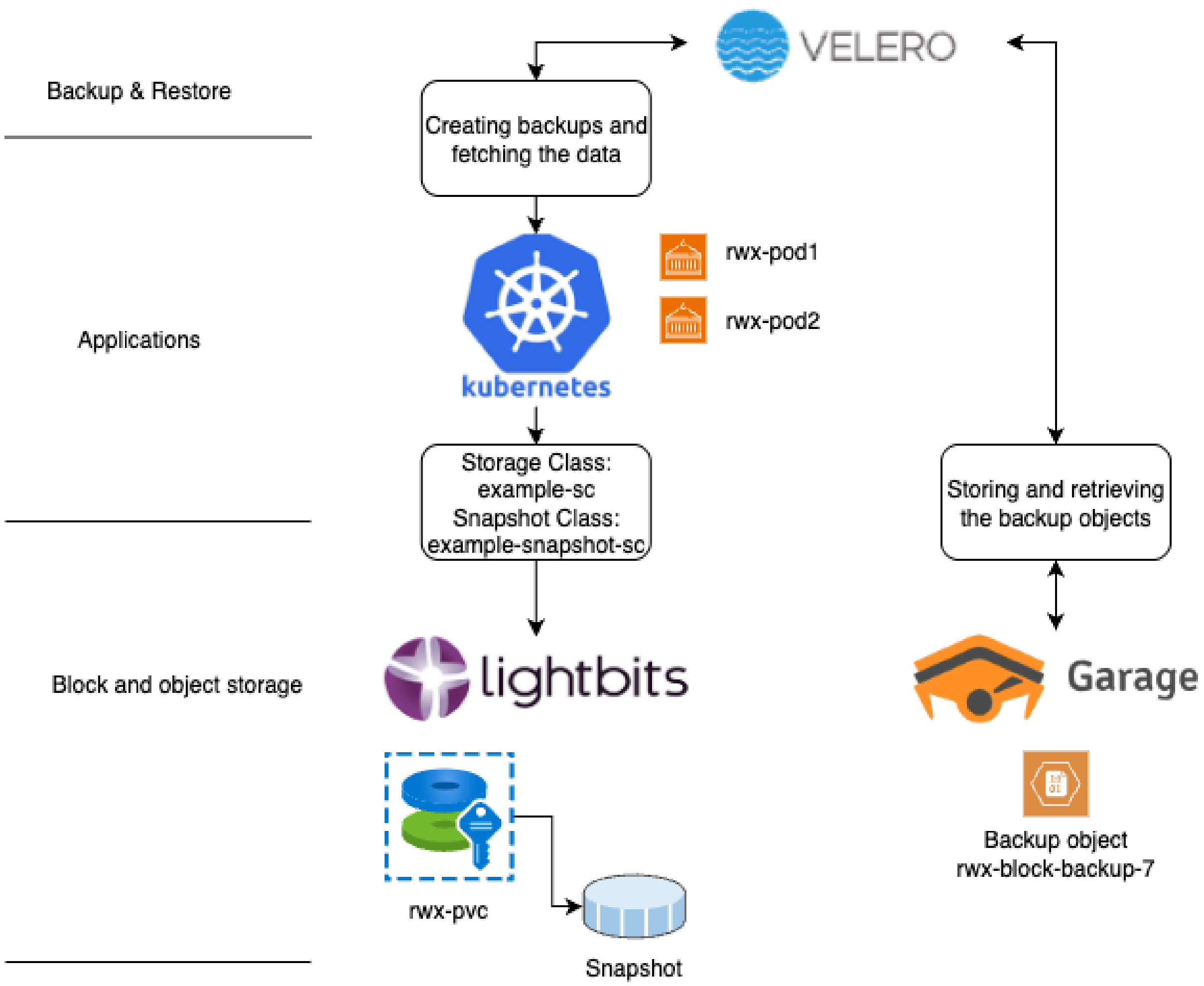
4. Pods and Their Associated PVC and Class

To understand how applications connect to both file systems and block volumes, it's essential to grasp the role of **Storage Classes** and **Snapshot Classes.** A Storage Class defines the properties of the underlying storage, determining whether an application will use a shared file system or a dedicated block volume. Meanwhile, a Snapshot Class governs the process of creating point-in-time copies of these volumes. The execution of these snapshots is managed by Kubernetes through CSI. Depending on the volume mode in Kubernetes for the PVC the snapshot will be created internally or externally on the storage, ensuring consistent and reliable backups.

The following is a diagram illustrating the filesystem example:



The following diagram is an illustration of the block volume example:



5. Required Changes for the Pod or PVC

To make backup creation easier, it is essential to configure labeling on the PVC and the pods, and to provide annotations for Velero for which plugin to use. For the filesystem, we created the `acme-fs-pod` label for the pod running in the namespace acme.

To apply the label and the annotation for the filesystem, adjust your yaml file for the pod with the new labeling and annotation:

Shell

```
metadata:
  name: "acme-fs-pod"
  annotations:
    backup.velero.io/backup-volumes: test-mnt
  labels:
    app: acme-fs-pod
```

To apply the label and the annotation for block, adjust your yaml file for the PVC with the new labeling and annotation:

Shell

```
metadata:
  name: rwx-pvc
  labels:
    app: rwx-pods
  annotations:
    velero.io/csi-backup-snapshot-class: "example-snapshot-sc" # Note this is
the snapshot class which is configure in kubernetes to create a snapshot on
Lightbits.
```

To apply the label and the annotation for block, adjust your yaml file for the pod with the new labeling for `rwx-pod1`:

Shell

```
metadata:
```

```
  name: "rwx-pod1"
  labels:
    app: rwx-pods
```

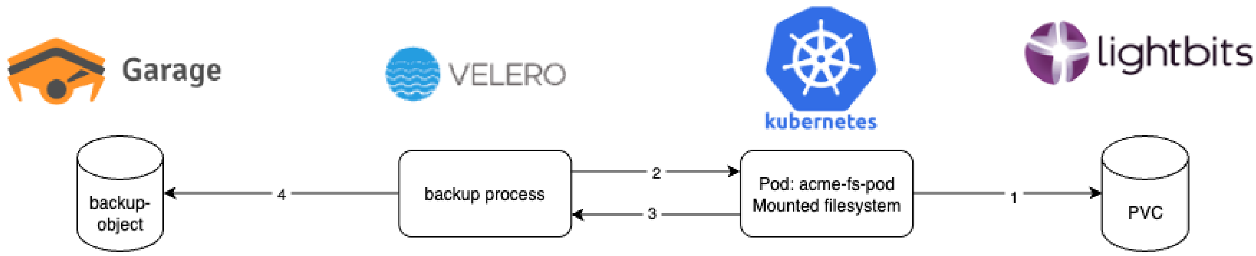
To apply the label and the annotation for block, adjust your yaml file for the pod with the new labeling for `rwx-pod2`:

```
Shell
metadata:
  name: "rwx-pod2"
  labels:
    app: rwx-pods
```

We have now created two app names for the backup and restore - namely acme-fs-pod and the combined rwx-pods. When we create the backup for the app rwx-pods, it will combine the external PVC and the mounted pods on it, as one backup target.

6. Back Up Kubernetes with volumeMode FileSystem

The diagram below illustrates what happens when we create a backup with Velero from a filesystem that is externally attached with a PVC.



Backup: Filesystem volumeMode in Kubernetes

1. PVC volume on Lightbits exposed to Kubernetes; Kubernetes mounts the PVC to a directory.
2. Velero requests a backup from Kubernetes.
3. Velero fetches all related information from Kubernetes.
4. Velero stores the backup information in the Garage s3 object store.

By using the app as a label, it is then easy to create a backup for `app=acme-fs-pod` or for `app=rwx-pods`.

Shell
```
velero backup create acme-fs-pod-backup-5 --selector app=acme-fs-pod
--include-namespaces acme
```

This creates the backup and an object in the Garage object store. To verify that the object was stored in Garage, we used the following command:

Shell
```
velero backup get acme-fs-pod-backup-5
NAME                   STATUS      ERRORS   WARNINGS   CREATED                          EXPIRES   STORAGE LOCATION   SELECTOR
acme-fs-pod-backup-5   Completed   0        3          2025-08-22 10:23:14 +0200 CEST   29d       default            app=acme-fs-pod
```

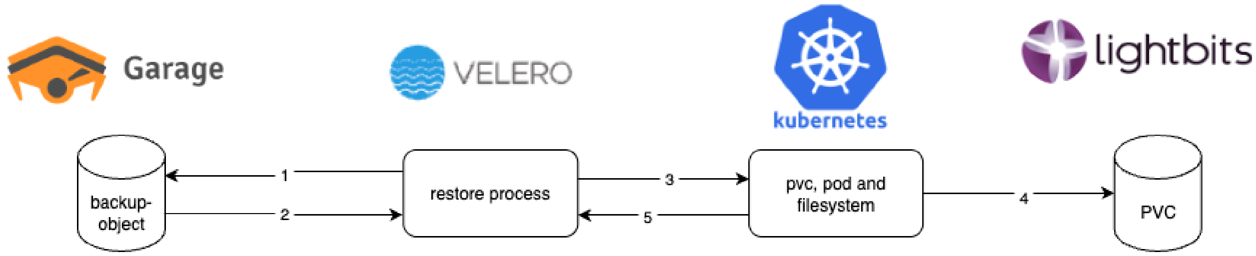
The following is the command to actually look inside Garage:

Shell
```
aws s3 ls s3://backup --recursive | grep acme-fs-pod-backup-5
2025-08-22 10:23:23         29
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-csi-volumesnapshotclasses.jso
n.gz
2025-08-22 10:23:23         27
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-csi-volumesnapshotcontents.js
on.gz
2025-08-22 10:23:23         29
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-csi-volumesnapshots.json.gz
2025-08-22 10:23:23         27
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-itemoperations.json.gz
2025-08-22 10:23:23       4634
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-logs.gz
2025-08-22 10:23:23        860
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-podvolumebackups.json.gz
2025-08-22 10:23:23        138
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-resource-list.json.gz
2025-08-22 10:23:23        148
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-results.gz
2025-08-22 10:23:23        368
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-volumeinfo.json.gz
2025-08-22 10:23:23         29
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5-volumesnapshots.json.gz
```

```
2025-08-22 10:23:23       3238
backups/acme-fs-pod-backup-5/acme-fs-pod-backup-5.tar.gz
2025-08-22 10:23:24       3257 backups/acme-fs-pod-backup-5/velero-backup.json
```

7. Restore Kubernetes with volumeMode FileSystem

The following diagram illustrates what happens when we restore a backup with Velero from Garage to a newly created PVC on external storage, and restore the pod mounted on that new PVC with a filesystem.



Restore: Filesystem volumeMode in Kubernetes

1. Velero requests the information about the backup information from Garage.
2. Velero fetches the restore information from Garage.
3. Velero pushes the restore information to Kubernetes.
4. Kubernetes creates the PVC on Lightbits, creates the pod, and mounts the file system in the pod.
5. Kubernetes informs Velero that the job is done.

Now we have our backup, and we need to do a restore. But before we do, we will first delete the pod and the PVC. The PVC was located on Lightbits, but needed to be created correctly on Lightbits as well (remember that the label was only called `app=acme-fs-pod`).

The following is the restore command we used:

```
Shell
velero restore create --from-backup acme-fs-pod-backup-5
```

To double-check that the PVC and pod are fully restored and up and running:

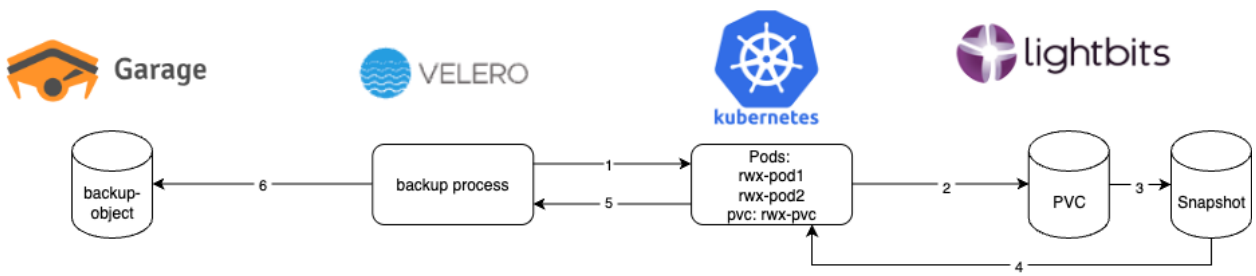
```
Shell
kubectl get pvc,pod -n acme --show-labels
NAME                                STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE    LABELS
persistentvolumeclaim/acme-fs-pvc   Bound
pvc-751584fc-2204-4a29-82ac-bda2ebd6ad2c   14Gi       RWO            acme-sc
<unset>                  106s
velero.io/backup-name=acme-fs-pod-backup-5,velero.io/restore-name=acme-fs-pod-b
ackup-5-20250822134022

NAME                 READY   STATUS    RESTARTS   AGE    LABELS
pod/acme-fs-pod   1/1     Running   0          106s
app=acme-fs-pod,velero.io/backup-name=acme-fs-pod-backup-5,velero.io/restore-na
me=acme-fs-pod-backup-5-20250822134022
```

When we look at the labels, it clearly shows that the backup and the store labels are added.

8. Back Up Kubernetes with volumeMode Block

The following diagram illustrates what happens when we create a backup with Velero from a block device that is externally attached with a PVC and running multiple pods on the same PVC, using an externally-created CSI snapshot.



Backup: Block volumeMode in Kubernetes

1. Velero requests a backup from Kubernetes.
2. Kubernetes requests a snapshot on Lightbits through CSI.
3. Lightbits creates the snapshot.
4. The snapshot is visible for Kubernetes.
5. Velero gets the information from Kubernetes.
6. Velero stores the backup information in the Garage s3 object store.

For the block mode backup operations, the command is actually the same:

Shell

```
 velero backup create rwx-block-backup-7 --selector app=rwx-pods
--include-namespaces default
```

What happens now is that the PVC running on a Lightbits volume will have a snapshot created by the CSI snapshot class from Lightbits. The PVC and the pods are backed up.

Verify the backup in Garage:

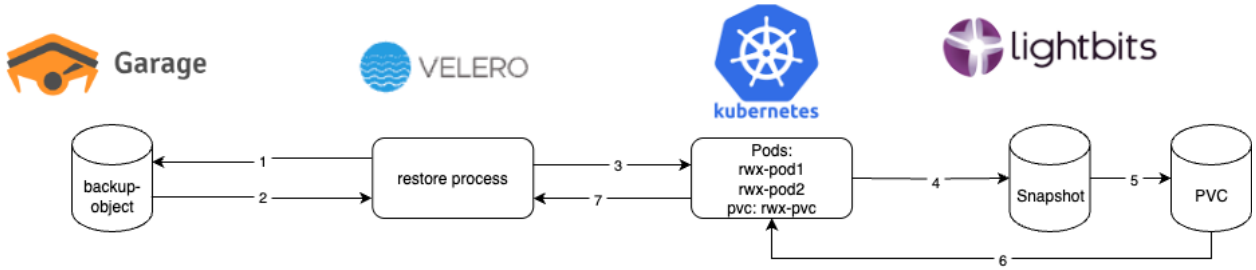
Shell

```
aws s3 ls s3://backup --recursive | grep rwx-block-backup-7

Output:
2025-08-26 11:53:53        426
backups/rwx-block-backup-7/rwx-block-backup-7-csi-volumesnapshotclasses.json.gz
2025-08-26 11:53:53       2396
backups/rwx-block-backup-7/rwx-block-backup-7-csi-volumesnapshotcontents.json.gz
2025-08-26 11:53:53       2088
backups/rwx-block-backup-7/rwx-block-backup-7-csi-volumesnapshots.json.gz
2025-08-26 11:53:53        977 backups/rwx-block-backup-7/rwx-block-backup-7-itemoperations.json.gz
2025-08-26 11:53:52       7611 backups/rwx-block-backup-7/rwx-block-backup-7-logs.gz
2025-08-26 11:53:53         29 backups/rwx-block-backup-7/rwx-block-backup-7-podvolumebackups.json.gz
2025-08-26 11:53:53        755 backups/rwx-block-backup-7/rwx-block-backup-7-resource-list.json.gz
2025-08-26 11:53:53        148 backups/rwx-block-backup-7/rwx-block-backup-7-results.gz
2025-08-26 11:53:55        464 backups/rwx-block-backup-7/rwx-block-backup-7-volumeinfo.json.gz
2025-08-26 11:53:53         29 backups/rwx-block-backup-7/rwx-block-backup-7-volumesnapshots.json.gz
2025-08-26 11:53:55      11908 backups/rwx-block-backup-7/rwx-block-backup-7.tar.gz
2025-08-26 11:53:55       3568 backups/rwx-block-backup-7/velero-backup.json
```

9. Restore Kubernetes with volumeMode Block

The diagram below illustrates what happens when we restore a backup with Velero from Garage to a newly created PVC on external storage, and restore the pods mounted on that new PVC with a filesystem by using the previously created external snapshot.



Restore: Block volumeMode in Kubernetes

1. Velero requests the information about the backup information from Garage.
2. Velero fetches the restore information from Garage.
3. Velero pushes the restore information to Kubernetes.
4. Kubernetes informs Lightbits to create a volume from the snapshot.
5. Lightbits creates the volume.
6. Kubernetes mounts the volume as a PVC, stores the pods on the PVC, and starts the pods.
7. Kubernetes informs Velero that the job is done.

We deleted the pods and the PVC and restored them with this command. Take a look at the labels as well.

The labels before the restore:

Shell
```
kubectl get pvc,pods --show-labels
NAME                                STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE   LABELS
persistentvolumeclaim/rwx-pvc   Bound
pvc-0cbd428c-bcd2-46b9-b220-6a19c680c638   20Gi      RWX            example-sc
<unset>                  76m   app=rwx-pods

NAME           READY   STATUS    RESTARTS   AGE   LABELS
pod/rwx-pod1   1/1     Running   0          76m   app=rwx-pods
pod/rwx-pod2   1/1     Running   0          76m   app=rwx-pods
```

The restore command:

Shell
```
velero restore create --from-backup rwx-block-backup-7
```

Even though the PVC has been deleted, the snapshot remained in Lightbits. This snapshot is tightly integrated with the restore capabilities of Velero. Without this snapshot, the restore will not function. The restore creates the PVC and restores the pods.

Take a look at the labels after restore:

```
Shell
kubectl get pvc,pods --show-labels
NAME                            STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE   LABELS
persistentvolumeclaim/rwx-pvc   Bound
pvc-e4e8b00e-7e46-4474-9be8-1e50ee77d399   20Gi       RWX            example-sc
<unset>                 6s
app=rwx-pods,velero.io/backup-name=rwx-block-backup-7,velero.io/restore-name=rw
x-block-backup-7-20250822135203,velero.io/volume-snapshot-name=velero-rwx-pvc-d
lc86

NAME             READY   STATUS              RESTARTS   AGE   LABELS
pod/rwx-pod1   0/1     ContainerCreating   0          6s
app=rwx-pods,velero.io/backup-name=rwx-block-backup-7,velero.io/restore-name=rw
x-block-backup-7-20250822135203
pod/rwx-pod2   0/1     ContainerCreating   0          6s
app=rwx-pods,velero.io/backup-name=rwx-block-backup-7,velero.io/restore-name=rw
x-block-backup-7-20250822135203
```

10. Restoring Just One Pod on the Shared PVC

If one of the pods is having problems and needs to be restored, we can simply delete the pod called `rwx-pod2` and restored the pod with the same command:

```
Shell
velero restore create --from-backup rwx-block-backup-7
```

The results are as follows:

```
Shell
kubectl get pvc,pods --show-labels
NAME                            STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE     LABELS
```

```
persistentvolumeclaim/rwx-pvc   Bound
pvc-e4e8b00e-7e46-4474-9be8-1e50ee77d399   20Gi       RWX            example-sc
<unset>                 4m30s
app=rwx-pods,velero.io/backup-name=rwx-block-backup-7,velero.io/restore-name=rw
x-block-backup-7-20250822135203,velero.io/volume-snapshot-name=velero-rwx-pvc-d
lc86

NAME           READY   STATUS    RESTARTS   AGE     LABELS
pod/rwx-pod1   1/1     Running   0          4m30s
app=rwx-pods,velero.io/backup-name=rwx-block-backup-7,velero.io/restore-name=rw
x-block-backup-7-20250822135203
pod/rwx-pod2   1/1     Running   0          3s
app=rwx-pods,velero.io/backup-name=rwx-block-backup-7,velero.io/restore-name=rw
x-block-backup-7-20250822135629
```

The `rwx-pod1` stayed running and was not touched; however, `rwx-pod2` was restored and started. When you look closely at the labels, you can see that `rwx-pod2` has a new time stamp; it was `20250822135203` and became `20250822135629`. The timestamp for `rwx-pod1` has not changed.

11. Conclusion

This comprehensive guide has demonstrated how the powerful combination of Velero, Garage, and Lightbits transforms Kubernetes data protection from a complex chore into a **seamless, automated process.** By integrating these leading technologies, organizations can establish a robust backup and restore solution that not only safeguards critical application data, but also simplifies disaster recovery and ensures business continuity.

The proven, step-by-step approach detailed in this paper highlights the synergy between **Velero's intelligent backup capabilities, Garage's scalable S3 object storage**, and **Lightbits' high-performance, efficient data platform.**

The result is an integrated solution that leverages external CSI snapshots for lightning-fast block-level backups and restores, as well as efficient file-system backups. Whether you're recovering an entire namespace or a single pod, the process is streamlined and reliable. The **ease of configuration** and the **demonstrated performance gains** of this architecture empower IT teams to focus on innovation rather than worrying about data loss.

By embracing this strategic data protection framework, businesses can confidently scale their Kubernetes environments, knowing that their data is secure, accessible, and ready for any challenge that lies ahead.

About Lightbits Labs

Lightbits Labs® (Lightbits) invented the NVMe over TCP protocol and offers best-in-class software-defined block storage that enables data center infrastructure modernization for organizations building a private or public cloud. Built from the ground up for low consistent latency, scalability, resiliency, and cost-efficiency, Lightbits software delivers the best price/performance for real-time analytics, transactional, and AI/ML workloads. Lightbits Labs is backed by enterprise technology leaders [Cisco Investments, Dell Technologies Capital, Intel Capital, Lenovo, and Micron] and is on a mission to deliver the fastest and most cost-efficient data storage for performance-sensitive workloads at scale.

www.lightbitslabs.com

info@lightbitslabs.com

US Offices
1830 The Alameda,
San Jose, CA 95126,
USA

Israel Office
17 Atir Yeda Street,
Kfar Saba 4464313,
Israel

LBWP16/2025/06