Using Cloud Services and Object Storage

Bacula Community Version

This document describes how to extend your local Bacula infrastructure with cloud storage capacities. Bacula is best suited to implement a cloud backup and restore strategy for you, because Bacula itself is already highly configurable and scalable. This flexibility allows Bacula to seamlessly scale further into the cloud. The techniques presented in this paper enable you to easily extend your storage infrastructure without having high initial investments in storage hardware.

Bacula Systems White Paper

Version 1.1, December 15, 2018

Contents

Glossary

- **AMI** Amazon Machine Image
- **AWS** Amazon Web Services: http://aws.amazon.com/
- **BEE** Bacula Enterprise Edition
- **CA** Certificate Authority
- **Ceph** Distributed object store and file system http://ceph.com
- **CLI** Command Line Interface
- **CPU** Central Processing Unit
- **DIR** Bacula Director
- **DNS** Domain Name Service (also: Domain Name System)
- **EBS** (Amazon EBS) Elastic Block Store http://aws.amazon.com/ebs/
- **EC2** (Amazon EC2) Amazon Elastic Compute Cloud http://aws.amazon.com/ec2/
- **FD** Bacula File Daemon (backup client)
- **FUSE** Filesystem in User Space
- **IQN** iSCSI Qualified Name
- **iSCSI** Internet Small Computer System Interface
- **NTP** Network Time Protocol
- **PKI** Public Key Infrastructure
- **RioFS** FUSE implementation to locally mount cloud object storage
- **S3** (Amazon S3) Simple Storage Service http://aws.amazon.com/s3/
- **S3FS** FUSE implementation to locally mount cloud object storage
- **SD** Bacula Storage Daemon (backup media server)
- **SSH** Secure Shell
- **TLS** Transport Layer Security
- **VM** Virtual Machine
- **VTL** Virtual Tape Library
- **VTS** (Amazon VTS) Virtual Tape Shelf

1 Introduction

This document was originally written for the Bacula Enterprise Edition customers, but it has been slightly modified to correspond to the Bacula Community version 7.4.x or later.

This document mainly references Amazon Web Services (AWS) in its illustrations of how to configure a remote Bacula Storage Daemon or how to sync local storage to the cloud. However, these concepts are presented in a way that can be transferred for integration with other cloud service providers.

The most straight forward approach is to create a cloud based virtual machine, install a Bacula Storage Daemon and add this storage target to your local running Bacula Director. Section 2 will illustrate this with an AWS EC2 Redhat 7 virtual machine with attached Amazon Elastic Block Store (EBS).

Sections 3 and 4 will demonstrate a technique to sync local storage into the cloud. This can also be reversed: assuming you have data stored in Amazon S3, you can sync it back to your local infrastructure and do backup with Bacula (see section 5).

If you do not wish to invest in any local storage space at all, you can directly mount cloud storage on your on-premise machines. Bacula itself needs a block storage device to write to. Therefore the use of an object store will always need a mapper. We experimented with RioFS and S3FS (see section 6), and we advise to test and carefully read their respective documentations as these tools may fit your specific needs and should be put in production with caution (at this point in time).

A note about costs

As you will know storing data in the cloud will create costs. Amazon for instance has a pricing model for each of its storage tiers and in addition the costs will vary with the region you use:

https://aws.amazon.com/s3/pricing/

Data transfer needs to be considered as well: While upload of data is typically free for all cloud storage providers, the download is charged which means that restores can become expensive. Also note, that when you write data to a volume with Bacula, some data will go the opposite direction for data verification and other important tasks.

2 Operate a remote Bacula Storage Daemon in the cloud

You will need an AWS account:

- https://aws.amazon.com/

2.1 Setup of the virtual machine

Sign in to the AWS console (https://aws.amazon.com/console/) and choose a region in the top menu (this way you can control where your data will be physically located, making it easy to meet regional compliance and data residency requirements, needed for some EU countries, see also the EU Data Protection Directive). Then go to *Services* → *Compute* → EC2. Click on *Launch Instance* and follow the wizard to create a virtual machine. Make sure to configure the *Instance Details*, add enough storage for your Bacula volumes, and define a *Security Group* with inbound rules including the Bacula SD port:

Type	Protocol	Port Range	Source
Custom TCP Rule	TCP	9103	W.X.Y.Z/32
SSH	TCP	22	W.X.Y.Z/32
All ICMP	All	N/A	W.X.Y.Z/32

Table 1: Inbound rules for AWS security group

The ICMP rule for ping is optional. In order to SSH into the machine you will have to create a key pair first (EC2 dashboard → *Key Pairs*). Define a single IP address as allowed source or a range of IP addresses. It is not recommended to leave the default setting (0.0.0.0/0), which allows all IP addresses to access your instance. Amazon also prints a warning regarding this.

In the Bacula Systems test lab we chose a Red Hat Enterprise Linux 7.2, type t2.micro, which comes with the free tier offering that Amazon gives new customers for the first twelve months of their subscription. It was enough for test purposes, but if you want to use this in a production setting, we recommend to have more CPU power in your virtual machine, because you will want to enable Bacula TLS encryption by default.

2.2 Security Considerations

All backup data that will be sent to your cloud SD will go though the Internet so you will definitely want to have some decent compression and you will want to

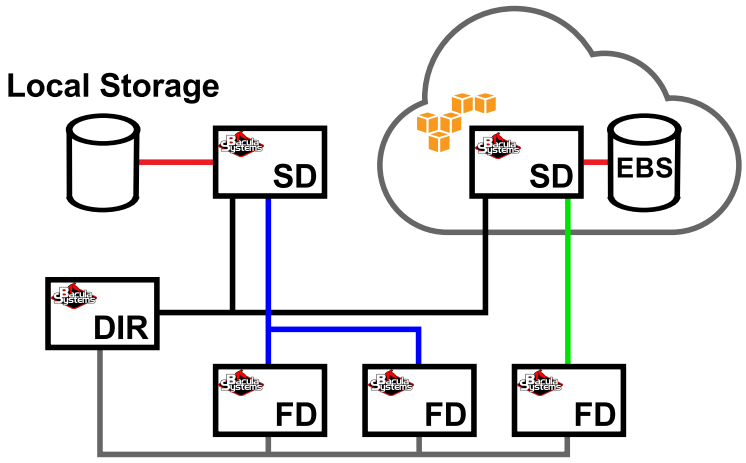


Figure 1: Setup of remote SD in the cloud

encrypt the channel. Read the *Bacula TLS* chapter in the main manual for detailed instructions.

In addition to encrypting the communications channel we highly recommend to activate data encryption with Bacula, which will put additional CPU load on your FD. Note that data encryption will only encrypt file contents, not file names. Example configurations for the FileSets and how to set up the necessary key pairs can be found in the *Data Encryption* chapter of the main manual.

You will need to have a PKI in place, i.e. a master key pair (CA) and signed certificates for all FDs that will backup to the cloud (for TLS) and additional key pairs for every FD that will encrypt the backup data.

2.3 Install and Configure the Bacula Storage Daemon

Please update the base system first (the Amazon Redhat image is not necessarily up to date):

```
[root@ip-172-31-24-139 ~]# yum update
```

Now add the Bacula Community repository in `/etc/yum.repos.d/`

```
[root@ip-172-31-24-139 ~]# cat /etc/yum.repos.d/bacula.repo
[Bacula]
name=Bacula Community Version
##### Note, this link does not yet work ######
baseurl=https://www.bacula.org.com/dl/repo-XXXXX/rpms/bin/7.4.4/rhel7-64/
enabled=1
protect=0
```

```
gpgcheck=0
[root@ip-172-31-24-139 ~]#
```

```
[root@ip-172-31-24-139 ~]# yum install bacula-postgresql
```

Make sure to disable the service for the DIR, you will not need it on this machine:

```
[root@ip-172-31-24-139 ~]# systemctl disable bacula-dir
```

You can even delete the configuration file (`/opt/bacula/etc/bacula-dir.conf`) for the DIR completely.

In our tests we added a second Amazon Elastic Block Store (EBS) volume to the VM, formatted it with the XFS filesystem, and created the mount point in `/srv/`:

```
[root@ip-172-31-24-139 ~]# cfdisk /dev/xvdb

Disk has been changed.

WARNING: If you have created or modified any
DOS 6.x partitions, please see the cfdisk manual
page for additional information.
[root@ip-172-31-24-139 ~]# mkfs.xfs -f /dev/xvdb
meta-data=/dev/xvdb              isize=256    agcount=4, agsize=655360 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=2621440, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
[root@ip-172-31-24-139 ~]# blkid /dev/xvdb
/dev/xvdb: UUID="fe653c94-5f53-44e5-8f9a-e52c83854540" TYPE="xfs"
[root@ip-172-31-24-139 ~]# vim /etc/fstab
[root@ip-172-31-24-139 ~]# cat /etc/fstab
#
# /etc/fstab
# Created by anaconda on Mon Nov  9 20:20:10 2015
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=379de64d-ea11-4f5b-ae6a-0aa50ff7b24d /                       xfs     defaults        0 0
UUID=fe653c94-5f53-44e5-8f9a-e52c83854540 /srv/storage            xfs     defaults        0 0
[root@ip-172-31-24-139 ~]# cd /srv/
[root@ip-172-31-24-139 srv]# mkdir storage
[root@ip-172-31-24-139 srv]# mount /srv/storage/
[root@ip-172-31-24-139 srv]# chown -R bacula:disk storage/
[root@ip-172-31-24-139 srv]# cd
[root@ip-172-31-24-139 ~]#
```

The configuration of the SD is similar to any other local Bacula Storage Daemon you operate. In this example we configured one (disk) autochanger with two devices:

```
Storage {
  Name = sd.cloudsd2
  SDPort = 9103
  WorkingDirectory = "/opt/bacula/working"
  Pid Directory = "/opt/bacula/working"
  MaximumConcurrentJobs = 20
}

Director {
  Name = dir.clouddir
  Password = "baculasd2"
}

Autochanger {
  Name = chgr.cloudsd2
  Device = dev.cloud1, dev.cloud2
  Changer Command = ""
  Changer Device = /dev/null
}

Device {
  Name = dev.cloud1
  Media Type = Cloud1
  ArchiveDevice = /srv/storage
  LabelMedia = yes
  RandomAccess = yes
  AutomaticMount = yes
  RemovableMedia = no
  AlwaysOpen = no;
  MaximumConcurrentJobs = 5
  Autochanger = yes
}

Device {
  Name = dev.cloud2
  MediaType = Cloud1
  ArchiveDevice = /srv/storage
  LabelMedia = yes;
  Random Access = yes
  AutomaticMount = yes
  RemovableMedia = no
  AlwaysOpen = no
  MaximumConcurrentJobs = 5
  Autochanger = yes
}

Messages {
  Name = Standard
  director = dir.clouddir = all
}
```

Note that the name in the Director resource needs to match the name of your (local) Bacula Director and the password needs to match the corresponding password in the Autochanger resource of your `bacula-dir.conf` for your Bacula Director (see below).

Start the `bacula-sd` service with

```
[root@ip-172-31-24-139 ~]# systemctl start bacula-sd
```

The service should be enabled by default, which a

```
[root@ip-172-31-24-139 ~]# systemctl status bacula-sd
```

should confirm.

2.4 Add the new storage target to your Bacula Director

The new Autochanger resource in your `bacula-dir.conf` should contain the DNS string of the EC2 instance in the Amazon cloud (note value for key `Address`). You can find this unique string in the *Instances* dashboard of the web interface in the lower panel after you have selected your VM.

Additional entry in `bacula-dir.conf`:

```
Autochanger {
  Name = chgr.cloudsd2
  Address = ec2-54-200-242-87.us-west-2.compute.amazonaws.com
  SDPort = 9103
  Password = "baculasd2"
  Device = chgr.cloudsd2
  MediaType = Cloud1
  MaximumConcurrentJobs = 10
  Autochanger = chgr.cloudsd2
}
```

After a *reload* of your Bacula Director configuration (or a restart of the service), please check with status storage if the cloud target can be accessed:

```
[root@clouddir ~]# /opt/bacula/bin/bconsole
Connecting to Director clouddir:9101
1000 OK: 10002 dir.clouddir Version: 8.4.10 (06 April 2016)
Enter a period to cancel a command.
*status storage
The defined Storage resources are:
     1: File1
     2: chgr.localsd1
     3: chgr.cloudsd2
Select Storage resource (1-3): 3
Connecting to Storage daemon chgr.cloudsd2 at ec2-54-187-147-64.us-west-2.compute.amazonaws.com:9103

sd.cloudsd2 Version: 8.4.10 (06 April 2016) x86_64-redhat-linux-gnu-bacula redhat Enterprise release
Daemon started 11-Apr-16 10:50. Jobs: run=1, running=0.
 Heap: heap=135,168 smbytes=399,830 max_bytes=742,957 bufs=105 max_bufs=123
 Sizes: boffset_t=8 size_t=8 int32_t=4 int64_t=8 mode=0,2010 newbsr=0
 Res: ndevices=2 nautochgr=1
```

```
Running Jobs:
No Jobs running.
====

Jobs waiting to reserve a drive:
====

Terminated Jobs:
 JobId  Level    Files      Bytes   Status   Finished        Name
====================================================================
    13  Full      1,122    9.653 M  OK       11-Apr-16 10:54 job.cloudfd1
====

Device status:
Autochanger "chgr.cloudsd2" with devices:
   "dev.cloud1" (/srv/storage)
   "dev.cloud2" (/srv/storage)

Device file: "dev.cloud1" (/srv/storage) is not open.
    Drive 0 is not loaded.
    Available Space=10.68 GB
==

Device file: "dev.cloud2" (/srv/storage) is not open.
    Drive 0 is not loaded.
    Available Space=10.68 GB
==
====

Used Volume status:
====

Attr spooling: 0 active jobs, 0 bytes; 1 total jobs, 303,845 max bytes.
====

*
```

After this confirmation that the storage target can be accessed, you are ready to start your first job and do a backup into the cloud.

3 Sync a Local Directory to Amazon S3

Amazon offers high-level S3 commands with the AWS Command Line Interface. This tool can be used to sync local Bacula volumes into the cloud. We will demonstrate this here by copying the Bacula volumes from our SD into an S3 bucket.

3.1 Create a Bucket in Amazon S3

Log into your AWS console (https://aws.amazon.com/console/), select the right region and choose *Services* → *Storage & Content Delivery* → S3. Create a bucket and give it a unique name. In our test scenario we created a bucket called *bsyssdsync* with one folder *volumes* inside it.

3.2 Install the AWS Command Line Interface

A guide which explains how to install the AWS Command Line Interface (CLI) can be found here:

- http://docs.aws.amazon.com/cli/latest/userguide/installing.html

We chose pip to install the AWS CLI. This worked despite the fact that our CentOS 6.7 SD only offered Python 2.6, which produced several warnings. Amazon recommends to have at least Python version 2.7.

Make sure that Python is installed:

```
[root@localsd1 ~]# python --version
Python 2.6.6
[root@localsd1 ~]#
```

The pip tool wasn't installed in our case which can be checked with:

```
[root@localsd1 ~]# pip --help
-bash: pip: command not found
[root@localsd1 ~]#
```

Installation is easy (but produces warnings with Python versions < 2.7):

```
[root@localsd1 ~]# curl -O https://bootstrap.pypa.io/get-pip.py
[root@localsd1 ~]# python get-pip.py
```

Once you have pip, you can install the AWS CLI and check if the installation was successful:

```
[root@localsd1 ~]# pip install awscli
[root@localsd1 ~]# aws help
```

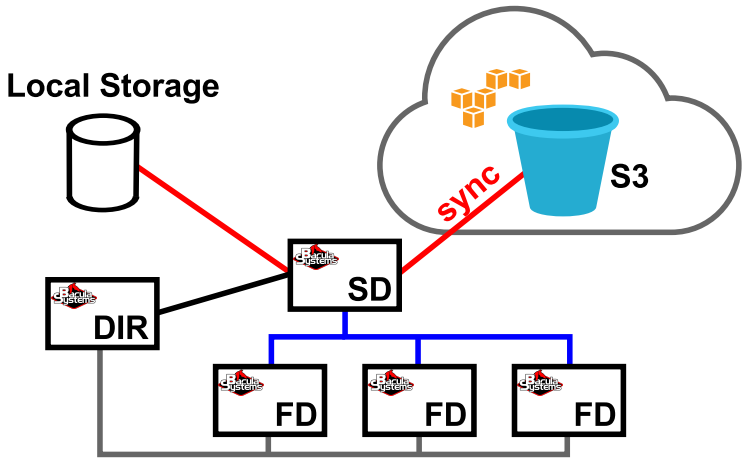


Figure 2: Syncing a local directory to an Amazon S3 bucket

3.3 Define user and export credentials

To be able to access your S3 buckets, you will need to create an authorized user in the web interface: *Services* → *Security & Identity* → IAM. Attach the policy *AmazonS3FullAccess* to this user. Create access keys in the web portal (tab *Security Credentials* in IAM) and export them as a CSV file.

Use the

```
[root@localsd1 ~]# aws configure
```

command to store the credentials locally (see also http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html)

3.4 Copy and sync files

You can list your Amazon S3 buckets with:

```
[root@localsd1 ~]# aws s3 ls
2016-04-11 21:13:02 bsyssdsync
[root@localsd1 ~]#
```

To copy volumes from your local SD to the cloud, use:

```
[root@localsd1 ~]# cd /srv/bacula-storage
[root@localsd1 bacula-storage]# ls
Vol-0002  Vol-0005
[root@localsd1 bacula-storage]#
```

```
[root@localsd1 bacula-storage]# aws s3 cp Vol-0002 s3://bsyssdsync/volumes/
upload: ./Vol-0002 to s3://bsyssdsync/volumes/Vol-0002
[root@localsd1 bacula-storage]#
```

This of course only makes sense when you have only one job per volume and your volumes are marked Full by Bacula after the job. You could trigger the upload to the cloud in a RunScript after the job and then delete the local copy. You would have to make sure though that in the restore case all volumes are available again in the local file system.

The aws s3 cp works in both directions:

```
[root@localsd1 ~]# aws s3 cp <source> <destination>
```

and behaves like the UNIX cp command (more details: http://aws.amazon.com/cli/). However, when you have more than one job per volume, and volumes with fixed maximum size configured in Bacula, you will want to sync the directory with Bacula volumes to S3. The AWS CLI has a command for that:

```
[root@localsd1 ~]# aws s3 sync /srv/bacula-storage s3://bsyssdsync/volumes
```

In this case you would identify Full volumes with a Bacula Catalog query and delete them after all backups have been run and the volumes have been synced. Again, you will need to make sure that they are available when you want to restore data.

When it comes to Bacula reusing volumes (after the configured retention times have passed), you would probably use a different configuration approach in a cloud scenario: Configure retention times to be indefinitely long (i.e. years), the volumes will be synced away into the cloud, deleted from disk, and then you will use Amazon mechanisms to tier the volumes to less expensive storage, from

General Purpose (Amazon S3 Standard)
↓
Infrequent Access (Amazon S3 Standard - Infrequent Access)
↓
Archive (Amazon Glacier)

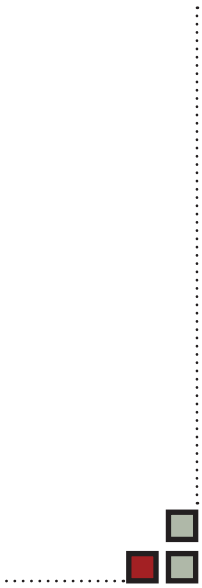
and finally delete them. Learn more about Amazon Storage Classes: https://aws.amazon.com/s3/storage-classes/. Policies can be set for each bucket independently in the AWS web portal.

3.5 Time Coordination

It seems that Amazon S3 syncing is sensitive to clock differences. If you get a S3-RequestTimeTooSkewed error during aws s3 sync you should use Amazon NTP servers:

http://www.emind.co/how-to/how-to-fix-amazon-s3-requesttimetooskewed

The NTP timing issue also cropped up with the second tool we tested (Rclone, see next section). Setting up NTP with Amazon servers (like described in the above link) helped in both cases.



4 Sync a local directory with Rclone

The Rclone tool is a very flexible way to sync to cloud based storage systems:

- http://rclone.org/

It not only supports Amazon S3, but also Google Drive, Dropbox, Microsoft One, Yandex Disk, and many others. You can even sync to or from your local Ceph cluster.

For this section we assume that you have an Amazon S3 bucket available (see subsection 3.1) and also a defined user with credentials (see subsection 3.3).

4.1 Install Rclone

The installation is very easy: download a package for your operating system from http://rclone.org/downloads/, unpack it and follow the few instructions in the provided README.txt file. Rclone is a program written in Go, and only consists of a stand-alone executable. In our case we performed the following steps for our CentOS 6 SD:

```
[root@localsd3 ~]# wget http://downloads.rclone.org/rclone-v1.29-linux-amd64.zip
[root@localsd3 ~]# unzip rclone-v1.29-linux-amd64.zip
[root@localsd3 ~]# cd rclone-v1.29-linux-amd64
[root@localsd3 rclone-v1.29-linux-amd64]# ls
rclone  rclone.1  README.html  README.txt
[root@localsd3 rclone-v1.29-linux-amd64]# cp rclone /usr/sbin/
[root@localsd3 rclone-v1.29-linux-amd64]# chown root:root /usr/sbin/rclone
[root@localsd3 rclone-v1.29-linux-amd64]# chmod 755 /usr/sbin/rclone
[root@localsd3 rclone-v1.29-linux-amd64]# mkdir -p /usr/local/share/man/man1
[root@localsd3 rclone-v1.29-linux-amd64]# cp rclone.1 /usr/local/share/man/man1/
```

Rclone has excellent documentation, and a very extensive man page. Further you will find dedicated web pages for the different cloud targets (Amazon S3: http://rclone.org/s3/). In our example we needed to configure Rclone with our Amazon user credentials. The README.txt file righfully says:

> First you'll need to configure rclone. As the object storage systems have quite complicated authentication these are kept in a config file .rclone.conf in your home directory by default. (You can use the --config option to choose a different config file.)
>
> The easiest way to make the config is to run rclone with the config option:
>
> rclone config

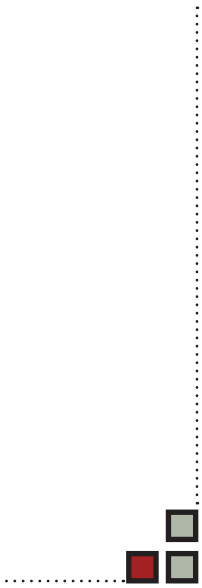
The configuration will lead you through several menus where you select your choices by pressing numbers (this is very similar to Bacula's bconsole menu). The Rclone application appears to be very mature, and after initial configuration we were able to list the content of a newly created Amazon S3 bucket, and sync the local Bacula storage directory to it:

```
[root@localsd3 ~]# rclone ls AWS:bsysnewsync

Transferred:            0 Bytes (   0.00 kByte/s)
Errors:                 0
Checks:                 0
Transferred:            0
Elapsed time:       700ms

[root@localsd3 ~]# rclone sync /srv/storage AWS:bsysnewsync
2016/04/20 08:35:42 S3 bucket bsysnewsync: Building file list
2016/04/20 08:35:42 S3 bucket bsysnewsync: Waiting for checks to finish
2016/04/20 08:35:42 Waiting for deletions to finish
2016/04/20 08:35:42 S3 bucket bsysnewsync: Waiting for transfers to finish
2016/04/20 08:35:55 S3 bucket bsysnewsync: Waiting for deletes to finish (during+after)

Transferred:      1551928 Bytes ( 104.90 kByte/s)
Errors:                 0
Checks:                 0
Transferred:            2
Elapsed time:       14.4s

[root@localsd3 ~]#
```

Rclone also has the rclone move subcommand that will automatically delete the local copy of your Bacula volumes after the successful sync.

4.2 A Note on Object Storage

The way we use sync commands with the AWS command line interface or with Rclone only addresses one object (in Amazon S3 called a bucket) at a time. Thus we do not really make use of the object nature of these cloud storages. True object storage support in Bacula Enterprise Edition has been released as a cloud plugin in version 8.8.0, at a later time that plugin will be released to the community

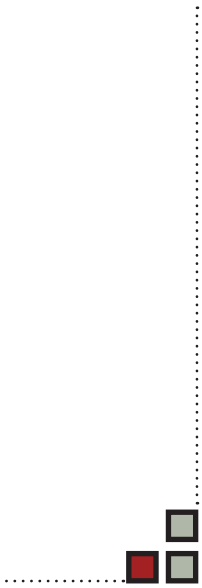


5 Backup Your S3 Data to Local Storage or Tape

The aws s3 cp or aws s3 sync commands that were introduced in section 3 and also the rclone sync command (see last section) work in both directions. This means that you can also sync files from the cloud to your local storage. Note that it takes significantly longer to upload data compared to download from Amazon S3.

The bidirectional capability enables you to backup data from the cloud to local storage or tape. You just need to have some local storage that you can sync to, and after copying the data you let Bacula backup your data to local storage or your tape library. You do not need to have the full amount of storage locally available. You can do the syncing in steps, which will save costs that you would have to spend otherwise on lots of local storage hardware.

The use of sync commands in the last two sections is not ideal, and many people have asked if it is possible to backup data to or from Amazon S3 directly without the need of local storage. This is also possible as you will see in the next section.



6 Cloud Storage as a Local Mount Point

The sync approach from the last sections has the disadvantage that you need to provide (at least some) local storage which produces additional costs. Wouldn't it be easier to directly mount Amazon S3 buckets to a local directory on your Bacula Storage Daemon? This is possible, but you should always keep in mind that Amazon S3 is an object storage system, but Bacula needs a block storage to write to and read from. Therefore we will need some kind of mapper to achieve a mount point that lets us directly access the S3 storage. And we shouldn't be surprised when we experience performance degradation due to sync processes.

6.1 Preparations

Please Note: We do not recommend RioFS or S3FS for production environments!

We will use RioFS (https://github.com/skoobe/riofs) to provide a local "mount point" for the S3 bucket. To install it on our CentOS 6 test system we had to run a few commands beforehand:

```
[root@localsd1 ~]# yum groupinstall "Development Tools"
[root@localsd1 ~]# yum install glib2-devel fuse-devel libevent-devel libxml2-devel openssl-devel
[root@localsd1 ~]# wget https://github.com/downloads/libevent/libevent/libevent-2.0.21-stable.tar.gz
[root@localsd1 ~]# tar xzf libevent-2.0.21-stable.tar.gz
[root@localsd1 ~]# cd libevent-2.0.21-stable
[root@localsd1 libevent-2.0.21-stable]# ./configure && make
[root@localsd1 libevent-2.0.21-stable]# make install
[root@localsd1 libevent-2.0.21-stable]# echo "/usr/local/lib/" > /etc/ld.so.conf.d/riofs.conf
[root@localsd1 libevent-2.0.21-stable]# ldconfig
[root@localsd1 libevent-2.0.21-stable]# export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
[root@localsd1 libevent-2.0.21-stable]# cd ..
[root@localsd1 ~]#
```

6.2 Install RioFS

Please Note: We do not recommend RioFS or S3FS for production environments!

Now we are ready to install RioFS:

```
[root@localsd1 ~]# wget https://github.com/skoobe/riofs/archive/master.zip
[root@localsd1 ~]# unzip master.zip
[root@localsd1 ~]# cd riofs-master/
[root@localsd1 ~]# ./autogen.sh
[root@localsd1 ~]# ./configure
[root@localsd1 ~]# make
[root@localsd1 ~]# make install
```

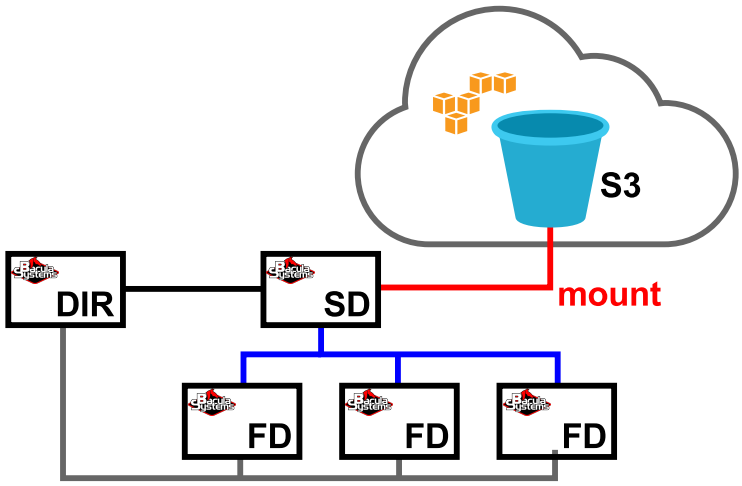


Figure 3: Mounting an S3 bucket as a local directoy

6.3 Mount S3 to a Local Directory

```
[root@localsd1 ~]# cat /root/.aws/credentials
[root@localsd1 ~]# export AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx
[root@localsd1 ~]# export AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
[root@localsd1 ~]# cd /srv
[root@localsd1 srv]# mkdir cloudmount
[root@localsd1 srv]# riofs --uid=497 --gid=6 -o "allow_other" bsyssdstorage cloudmount
```

6.4 Warnings

There are a few things to note here:

- The `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are the same as needed for the aws and rclone commands and can be found in the Amazon web portal (IAM section) and exported as a CSV file.
- Do not forget to set --uid and --gid here. Otherwise the folder permissions will be root:root and cannot be changed (this is a FUSE mount). We tried to let the bacula user do the RioFS mount (activate login shell, add to fuse group), but this did not work out.
- Mounting buckets failed at first, when we wanted to use an Amazon data center in Frankfurt (Germany). We learned that buckets in newer (post 2014) regions like Frankfurt use v4 signing and RioFS supports only v2 signing at this time. We haven't yet found documentation on how to fix this. The RioFS project is under active development, and this glitch will hopefully be

fixed soon. We side tracked to a data center in North America and this bucket we could sucessfully integrate.

- Backups worked, but terminated with a status of 'OK – with warnings'. After creating and labelling a new volume, Bacula could not immediately write to it:

  ```
  Error:  block.c:312 Write error at 0:0 on device "dev.file1"
  (/srv/cloudmount/volumes).  ERR=No such file or directory.
  ```

  Bacula created another new volume and then the write operations worked. This behavior is reproducible. However, the restore of the data was successful in the end.

6.5 S3FS Instead of RioFS

The mount point with RioFS was not completely stable (see warnings above) and the installation procedure where you have to compile the tool looks very experimental. Such an install process is not what you really want from a backup solution. You want it mature, stable and reliable. You do not want to worry about your data or your backups.

Therefore we tried a second tool called S3FS. The installation was similarly experimental (see http://tecadmin.net/mount-s3-bucket-centosrhel-ubuntu-using-s3fs/ for details), but in the end we were able to mount the remote S3 bucket and could see the Bacula volumes in our local mount point. But that was it, backups and restores in Bacula did not work at all. Bacula created hundreds of volumes in an attempt to write to the mount point, couldn't open a given volume after creation, and marked it with Error state after a timeout, then created the next volume. The backup job had to be cancelled.

The experience we have made with S3FS and RioFS is confirmed by several users on the Bacula Users Mailing list (see links below). To mount object storage locally is not as reliable as it should be for backup purposes.

Please Note: We do not recommend RioFS or S3FS for production environments!

- https://sourceforge.net/p/bacula/mailman/message/34559409/
- https://sourceforge.net/p/bacula/mailman/message/34194862/
- https://sourceforge.net/p/bacula/mailman/message/34118186/

7 Use AWS Storage Gateway VTL as a Target for Bacula

Amazon offers a cloud based Virtual Tape Library (VTL) as part of their AWS Storage Gateway. To use this VTL, you will need to run a gateway appliance either on-premises (VMware and Hyper-V images are provided by Amazon), or in AWS as an Amazon EC2 instance. For this white paper we went with the second option since we had an AWS account available.

In addition, you need a Bacula Storage Daemon, that connects to the iSCSI devices of the gateway. Since we chose to run the gateway in EC2, we had to run the SD also in EC2, because for gateways that are deployed on an Amazon EC2 instance, accessing the gateway over a public Internet connection is not supported. The elastic IP address of the Amazon EC2 instance cannot be used as the iSCSI target address.

7.1 Deploy the gateway virtual machine

Click on *Services* → *Storage & Content Delivery* → *Storage Gateway* and choose *Deploy a new gateway on Amazon EC2*.

In the upcoming screen make sure to choose the Gateway Type *Virtual Tape Library* in the lower part of the window. Do not click on *Proceed to Activation* but follow the four steps outlined in the upper part of the window first.

In step 1 you choose the AWS Storage Gateway AMI. Amazon recommends the Manual Launch option. You will receive a confirmation email that you have accepted the terms of usage. After that you can launch an instance in one of the available regions. Click on *Launch with EC2 Console*.

Note: AWS Storage Gateway only supports the m3, i2, c3, c4, r3, d2, and m4 instance types. The instance size must be at least type "xlarge". This means that you cannot use the VTL gateway as part of the free tier offer.

The next steps are very similar to the process for setting up the Amazon EC2 VM for the SD in subsection 2.1: Continue to follow the EC2 Launch Wizard as described in the Amazon documentation: add additional storage volumes for the gateway to use as cache storage and an upload buffer, tag the instance (optional), configure the Security Group (this is an important step, because certain ports need to be opened for local AWS network, communication with Amazon DNS server, and to connect to iSCSI targets exposed by the gateway).

Review and lauch the instance, and once it has finished initialization, copy the IP address from the Description tab on the EC2 Instances Overview page, and fill it into the Activation window that should still be open in a different browser window. Click on *Proceed to Activation*, and specify region, timezone and gateway name on the next page. As Medium Changer Type choose *AWS-Gateway-VTL*, and as Tape Drive Type choose *IBM ULT3580-TD5*.

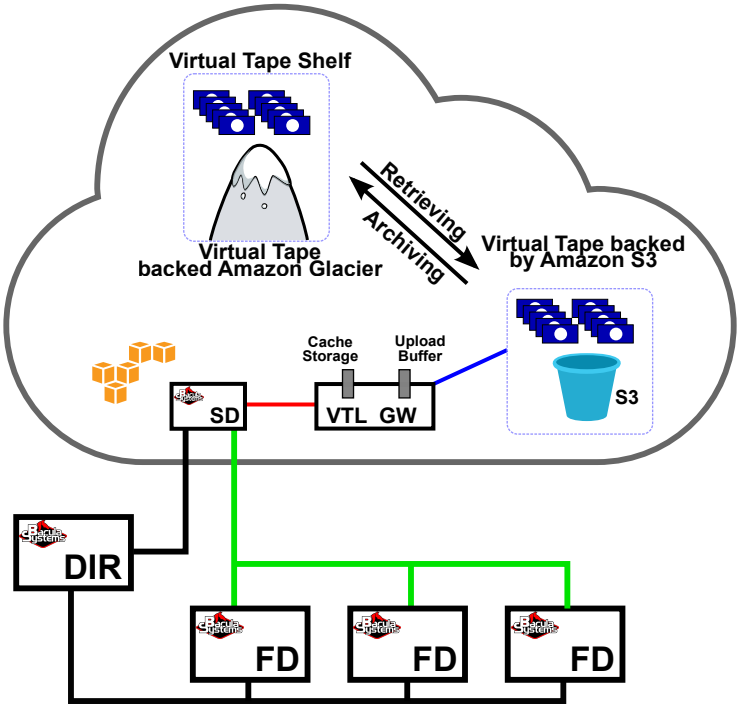


Figure 4: Amazon VTL gateway with S3 backed tapes and Amazon Glacier

After you have successfully activated the gateway it appears in your AWS Storage Gateway console (VTL Gateways) where you can create tapes with a click on a button. You will have to specify which of the attached EBS volumes you want to use for Upload Buffer and Cache Storage

7.2 Install Bacula Storage Daemon

Here we performed the same steps as in described in subsection 2.3. We chose again a t2.micro machine and configured SSH access.

Before installing the Bacula Storage Daemon a few steps are needed to connect to the gateway machine and configure the iSCSI initiator:

```
[root@ip-172-31-30-186 ~]# yum update
[root@ip-172-31-30-186 ~]# yum install iscsi-initiator-utils lsscsi mt-st mtx
[root@ip-172-31-30-186 ~]# systemctl start iscsid
[root@ip-172-31-30-186 ~]# iscsiadm --mode discovery --type sendtargets \
                           --portal PUBLIC_GW_IP:3260
[root@ip-172-31-30-186 ~]# iscsiadm --mode node \
                           --targetname iqn.1997-05.com.amazon:sgw-8c0ceee5-mediachanger \
                           --portal LOCAL_NETWORK_IP:3260,1 --login
```

You get the local network IP from the discovery command. The IQN in this example was for the media changer. Repeat the last step for all devices that you wish to initiate. After that you will be able to see them:

```
[root@ip-172-31-30-186 ~]# lsscsi -g
[2:0:0:0]    mediumx AWS      Gateway-VTL      0100  /dev/sch0  /dev/sg0
[3:0:0:0]    tape    IBM      ULT3580-TD5      0103  /dev/st0   /dev/sg1
[4:0:0:0]    tape    IBM      ULT3580-TD5      0103  /dev/st1   /dev/sg2
[5:0:0:0]    tape    IBM      ULT3580-TD5      0103  /dev/st2   /dev/sg3
[6:0:0:0]    tape    IBM      ULT3580-TD5      0103  /dev/st3   /dev/sg4
[7:0:0:0]    tape    IBM      ULT3580-TD5      0103  /dev/st4   /dev/sg5
[8:0:0:0]    tape    IBM      ULT3580-TD5      0103  /dev/st5   /dev/sg6
[9:0:0:0]    tape    IBM      ULT3580-TD5      0103  /dev/st6   /dev/sg7
[10:0:0:0]   tape    IBM      ULT3580-TD5      0103  /dev/st7   /dev/sg8
[11:0:0:0]   tape    IBM      ULT3580-TD5      0103  /dev/st8   /dev/sg9
[12:0:0:0]   tape    IBM      ULT3580-TD5      0103  /dev/st9   /dev/sg10
[root@ip-172-31-30-186 ~]#
```

In a production setup you will not want to rely on the device names, because they are not preserved when you reboot. A udev rule will give the media changer a predefined static device name, type 8 (=100) devices are media changers. For our testing purposes we stayed with `/dev/sg0`.

Now we can use mtx to see the status of the changer and load/unload a tape:

```
[root@ip-172-31-30-186 ~]# mtx -f /dev/sg0 status
```

The output of the status command is long in this case, because you have 10 drives and 3200 slots available (1600 regular slots and 1600 import/export slots, used when archiving and retreiving tapes to/from the Amazon Virtual Tape Shelf). So better pipe it into a pager:

```
[root@ip-172-31-30-186 ~]# mtx -f /dev/sg0 status | less
```

If you have created a tape in the Amazon Storage Gateway Management Console, you should see it in slot 1601. Use mtx to load it into a drive:

```
[root@ip-172-31-30-186 ~]# mtx -f /dev/sg0 load 1601 0
Loading media from Storage Element 1601 into drive 0...done
[root@ip-172-31-30-186 ~]#
[root@ip-172-31-30-186 ~]#
[root@ip-172-31-30-186 ~]# mtx -f /dev/sg0 status | head
  Storage Changer /dev/sg0:10 Drives, 3200 Slots ( 1600 Import/Export )
Data Transfer Element 0:Full (Storage Element 1 Loaded):VolumeTag = BSYSDC7E79
Data Transfer Element 1:Empty
Data Transfer Element 2:Empty
Data Transfer Element 3:Empty
Data Transfer Element 4:Empty
Data Transfer Element 5:Empty
Data Transfer Element 6:Empty
Data Transfer Element 7:Empty
Data Transfer Element 8:Empty
[root@ip-172-31-30-186 ~]#
```

With the `unload` command you can unload the tape, but in the end you will want to modify Bacula's mtx-changer script to do loading and unloading automatically for you. You will have to tweak a little bit to keep tapes in a drive until they are Full, because a tape that gets unloaded will be transferred to AWS immediately. If you want to retrieve it, additional costs apply.

7.3 SD configuration

`bacula-sd.conf`:

```
[root@ip-172-31-30-186 ~]# cat /opt/bacula/etc/bacula-sd.conf
Storage {
  Name = sd.cloudsd4
  SDPort = 9103
  WorkingDirectory = "/opt/bacula/working"
  PidDirectory = "/opt/bacula/working"
  MaximumConcurrentJobs = 20
}

Director {
  Name = dir.clouddir
  Password = "VTL"
}

Autochanger {
  Name = "VTLchgr"
  Device = VTLDrive1
  ChangerCommand = "/etc/bacula/scripts/mtx-changer %c %o %S %a %d"
  ChangerDevice = /dev/sg0
}

Device {
  Name = VTLDrive1
  Drive Index = 0
  MediaType = ULT3580-TD5
  ArchiveDevice = "/dev/tape/by-path/ip-172.31.28.41:3260-iscsi-iqn.1997-05.com.amazon:sgw-8c0ceee5-tapedrive
  RemovableMedia = yes
  RandomAccess = no
  AutoChanger = yes
}

Messages {
  Name = Standard
  director = dir.clouddir = all
}
```

Note that the Changer device in the Autochanger{} resource and the Archive Device in the Device{} resource need to be changed to your IDs respectively.

7.4 DIR configuration

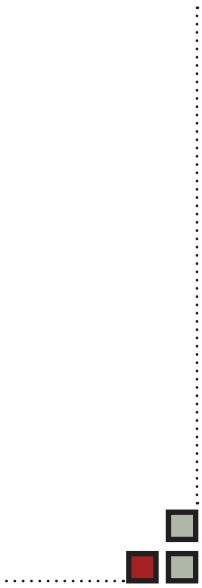
On the (local) director you will add the new storage target with the following Bacula resources:

```
Autochanger {
  Name = chgr.cloudsd4
  Address = ec2-54-93-63-125.eu-central-1.compute.amazonaws.com
  SDPort = 9103
  Password = "VTL"
  Device = VTLchgr
  MediaType = ULT3580-TD5
  MaximumConcurrentJobs = 10
  Autochanger = chgr.cloudsd4
}
```

The address for the SD in the cloud needs to be changed. We also defined a new pool for the VTL tapes:

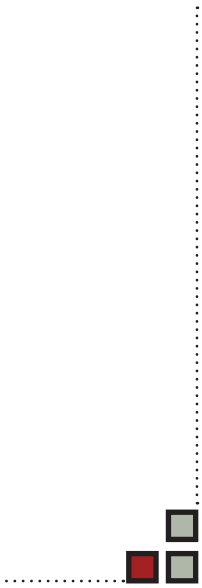
```
Pool {
  Name = pool.VTL
  PoolType = Backup
  Recycle = yes
  AutoPrune = yes
  VolumeRetention = 2 years
  MaximumVolumes = 100
}
```

Now you only need to create a job that will use this newly defined pool and storage, and you will be able to use your cloud based VTL.



For More Information

For more information on Bacula Enterprise Edition, or any part of the broad Bacula Systems services portfolio, visit www.baculasystems.com.



V. 1.1
Author(s): DOG