

How to Set Up an Apple Time Machine

BY BENEDICT REUSCHLING

Apple's time machine has been around for a number of years to back up Macs over the network. It is a simple solution to set up and runs in the background without bothering the user too much. Users can retrieve their files from previous versions of their file system or even restore their whole computer to a new one to continue where they left off.

Apple initially offered separate hardware called time capsules for this task, but in recent years focused more on backing up to their paid iCloud solution.

Some users may not want to trust their file system contents to some other computer out there beyond their reach and control. Luckily, setting up a FreeBSD system to act as a time capsule on the local network is still supported. In this article, we'll walk you through such a solution.

A time capsule is basically a service listening on incoming connections from the time machine protocol and then stores the data submitted (the latest backup delta) on the local file system. Fans of OpenZFS trust that the built-in data integrity features keep their data intact, which is why we're combining time machine with OpenZFS here. As we are trusting our most valuable files from the Mac to our FreeBSD system, we want to ensure that we can retrieve it again one fateful day when we desperately need it back.

Another consideration is noise and power use. Since this local time capsule system is supposed to be running 24/7 for backing up and retrieving the files, we don't want a noisy solution that also draws a lot of power. Surely, one could limit the service to only run during the hours when we are actually awake and use our Macs. Some people keep a separate time machine at work for redundancy reasons, so that could be limited to the typical 9 to 5 work hours. Nevertheless, we don't want to increase our energy bill too much. By the same token, running a time machine at home or in the office under our desks with noisy fans will disturb colleagues and family members alike. The solution to both problems is to use an ARM embedded board for the task. Not only are they cheap (cost-wise), but also come in a small form fac-

Luckily, setting up a FreeBSD system to act as a time capsule on the local network is still supported

tor which does not take up much space like a big server would. Pretty much all of them come without fans and are practically noiseless when running. Since ARM focused their chip development on energy efficiency, you'd be surprised how few Watts are needed to juice these boards. Finally, you don't need much computing horsepower to run a time machine server, as it mostly does I/O. I should also mention the cost factor: buying a small ARM board plus some external storage should still be cheaper than buying a time capsule from Apple. Maybe you'd like to donate some of the money saved this way to a BSD Foundation of your choice to support the continued development of the operating system?

I have run a time machine backup on a Raspberry Pi 3 with external storage connected to it for a while without any issues. You can build this solution on any recent ARM board that FreeBSD supports (i.e., boots and can install packages), it does not necessarily have to be a Raspberry Pi. The required configuration is not too complicated and once it is running, you can forget about it as it does not need constant attention. As long as you have some external storage, you can start making backups to it from your Mac. The reason why you want to use an external storage is that the flash on the boards is typically limited in capacity and may wear out when constantly written to. You can start with a single external disk and later create a ZFS mirror out of it when the next paycheck arrives. When the disk space gets low in your FreeBSD time capsule, the time machine protocol automatically removes older backups to make space. No intervention is necessary, it all runs on its own.

In this article, we assume that you have the board running with FreeBSD, connected to the network and external storage. It should be powerful enough to run ZFS on it, as this is my preferred solution. It will run just fine with UFS, so use that if your board's hardware is not strong enough for ZFS. Create a ZFS pool as outlined in the FreeBSD handbook to get started. We'll later create the necessary datasets on it as part of the setup.

First, we need to install two primary packages with dependencies that provide the time machine service:

```
# pkg install netatalk3 avahi-app
```

Avahi allows the discovery of the time machine service on the local network in an easy way. The Apple file server protocol, Apple Talk in version three, is what time machine is built on. Together, they will make the configured time capsule available on the network so that Macs can find and automatically back up their data to it. The installation should not take too long, depending on how powerful your ARM server is (or any kind of server you run this on). You can also decide to run the service in a jail, dedicating a jailed dataset to it for the backups. I'll leave that as an exercise for you to keep this article simple enough.

The backup files from the Mac should be stored for each individual user in their own directory. This way, we keep them separate from each other and allow other people to make backups as well. Let's assume our ZFS pool is called backup (what's in a name?). The following command creates a new dataset to hold all users' backups in their own directories:

```
# zfs create backup/timemachine
```

We also set some zfs options in case they are not set on the pool level already.

```
# zfs set atime=off backup/timemachine
# zfs set refquota=1T backup/timemachine
# zfs set reservations=1T backup/timemachine
# zfs set compression=zstd backup/timemachine
```

The first option disables the file system access time, which we don't need to run time machine successfully. The `refquota` and `reservation` options ensure that only 1 TB of pool storage will be allocated for the backups, but that they are guaranteed to be available no matter how much space non-timemachine files on the pool take up. Adjust this to fit your pool size and needs. Don't set this too low, though, or you won't be able to backup much from your Macs and older backups get deleted more often. The final option activates compression on the dataset. Be mindful of the compression algorithm set in the last option. Your ARM board may not be able to provide that much CPU power for the compression, so change this to a different algorithm or disable compression altogether. On my time machine, the compression ratio is low (1.01x currently), but it may depend on the type of files you back up from the Mac and how well they can be compressed.

Next, let's create a group called `timemachinists` that are allowed to use the time machine service. You don't want your noisy neighbor using your time machine for his backups, but maybe allow a new colleague in your office to back up her Mac, too.

```
# pw grouadd timemachinists
# pw groumod timemachinists -m bcr
```

Check the result of this operation using `pw groupshow timemachinists`. I'm the only user in that group at the moment. You can also pick a different group name as long as you reference it from the config file we'll show below. Each user should have their own dataset, so let's create one for myself and set proper permissions:

```
# zfs create backup/timemachine/bcr
# chown bcr:timemachinists /backup/timemachine/bcr
# chmod 0700 /backup/timemachine/bcr
# chmod 0777 /backup/timemachine
```

I allow only myself access to the `bcr` dataset, only the other `timemachinists` group members should be allowed to peak into my precious backups. Although the files are stored in a database format and not as they are on my Mac, I'm not taking any chances. On the `/backup/timemachine` dataset, the permissions are wider open for the service to properly access it. Now let's see how we reference this group and the mount point from the time machine configuration file. It is located in `/usr/local/etc/afp.conf` and contains the main time machine settings. To get started, the following configuration changes should be made:

```
[Global]
afp listen = 10.20.30.40
uam list = uams_dhx.so,uams_dhx2.so
mimic model = TimeCapsule6,116
disconnect time = 1
unix charset = UTF8
cnid scheme = dbd
file perm = 0640
directory perm = 0750
hostname = "Time Machine"
hosts allow = "10.20.30.0/24"
zeroconf = yes
log file = /var/log/afp.log
```

```
log level = default:info
vol preset = TimeMachine
vol dbpath = /var/netatalk/CNID/$u/$v/
```

```
[TimeMachine]
path = /backup/timemachine/$u
time machine = yes
valid users = @timemachinists
```

Here is what the options do, line by line, starting in the Global section. The “afp listen” line defines the host name or IP address of the machine that the service runs on and listens for incoming connections. This is the address that users will configure in the time machine settings on their Macs later. We chose 10.20.30.40 as an example here, adjust it to fit your own local network.

The “uam list” refers to the user access methods allowed. The ones we use here are the default ones that allow Diffie-Hellman eXchange protocol (versions 1 and 2) encoded passwords. Other possible options allow guest access (undesirable for most people) and Kerberos V, which may be interesting in a corporate setting.

If you care about what icon is displayed on your connecting Macs, the “mimic model” option is for you. If you’re feeling nostalgic, you can display a PowerBook (which predated time machine by some years) here. However, the Time Capsule option makes the most sense to use here.

A more useful option is the “disconnect time”. It may happen that connections get interrupted, and the time machine service keeps the session open, preventing further connection attempts with a “volume in use” error message. This option cleans up disconnected sessions after 1 hour. Adjust this if you get this error often, but you should not encounter it much when using the setting used here.

Specifying the server character set to the default UTF8 is done by the “unix charset” option. This should only be changed when you’re certain that you need it, otherwise leave this option alone.

A “cnid scheme” is what is being used for the backend of the volume. This is the database that keeps track of what files have been backed up, if and when they changed, and other administrative information. You don’t have to know much about this to run the time machine service, keeping the default dbd is fine for most people.

Both the “file perm” and “directory perm” define with what kind of permissions the files and directories from the connecting clients are stored, respectively. This could be more restrictive than the original permissions or less, but both cause more headaches than you have painkillers for, so leave the ones we suggested here.

The “hostname” parameter is the description of your time machine that the Mac users will see when clicking on the time machine icon in their status bar when correctly configured. Pick-

Although the files are stored in a database format and not as they are on my Mac, I’m not taking any chances.

ing a funny description here results in status messages between backups like “Last backup on black hole.” You’re easily amused, aren’t you?

Remember your noisy neighbor that should not be allowed to use your precious time machine storage for his Mac? The “hosts allow” option limits the service to certain hosts or networks, restricting everyone else. As much as your colleagues in the surrounding offices may complain, only the people sharing the same four walls around you can back up if you set it properly.

We want to use “zeroconf” to ease our burden to manually help computers find the service, so we set this option to yes.

Logging activity of the time capsule is a good idea to debug problems users may encounter when trying to use the service. The “log file” specifies the location of the log file and the “log level” the detail and number of messages being logged. The ones we use here are fine for day-to-day operations, while also not wearing out your board’s storage with too many writes. When debugging an issue, temporarily set it to warn or error for more details and restart the service.

With “vol preset” we define a section in the same configuration file for settings concerning the volume. Multiple volumes can be configured this way in the same configuration file, but for our purposes, a single one is enough.

The last option in the General section is “vol dbpath”. Remember our different users we may have tapping this service in the future? With this setting, each user gets their own independent settings in a subdirectory. My own volume may be called “bcrvol” and the settings for it get stored under `/var/netatalk/CNID/bcr/bcrvol/` (replacing the variables for my username and volume, respectively). The benefit of separating these by user and volume and not having one directory for all users is that if things get corrupted, this is limited to only one user. This will not happen often, but better safe than sorry (especially when it comes to backups).

We’re still not done yet with this file, but the last options are straightforward enough. The section we reference here from the [Global] section deals with who is able to access and where they should be allowed.

A volume “path” is where a backup from a certain user is stored. Since we don’t know who this will be down the road and adjust this file each time a new user comes along, we use the `$u` variable here. This way, my own files end up in `/backup/timemachine/bcr` and correspond to the datasets we created earlier. Don’t get confused here: the path is the file system path where the backed up files from the Mac will later reside. The “vol dbpath” option that used a similar variable is where the administrative database with meta information about the backup is stored. Even better, you don’t have to visit both directories at all and can forget about them once the service runs.

The “valid users” line specifies users or, in our case, the timemachinists group (distinguished from users by a leading @) that are allowed access. See how flexible this is? In the future, when we want to allow Susan Sunshine to also backup her Mac, we just need to create a user for

Logging activity of the time capsule is a good idea to debug problems users may encounter when trying to use the service.

her, add it into the timemachinists group, and create a dataset under `/backup/timemachine` with proper permissions for her.

```
# pw groupmod timemachinists -m susan
# zfs create backup/timemachine/susan
# chown susan:timemachinists /backup/timemachine/susan
# chmod 0700 /backup/timemachine/susan
```

No need to revisit this configuration file again, because variables and the group definitions will pick up the new user automatically. In case you still need to, each configuration line is described in more detail in `afp.conf(5)`. Let's save and close this file.

Just as Darth Vader felt a tremor in the Force in the presence of his old master, we want to automatically notify the Mac clients of the presence of your time machine service using Avahi. To that end, we create a new file in

```
/usr/local/etc/avahi/services/afp.service
```

and add the following contents:

```
<?xml version="1.0" standalone='no' ?><!--*-nxml-*-->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">%h</name>
  <service>
    <type>_afpovertcp._tcp</type>
    <port>548</port>
  </service>
</service-group>
```

This basically defines where the AFP protocol will listen on (the "afp listen" line from `afp.conf` plus the port 548 defined here). With this file in place, we only need to enable and start the services to finish the server side of the setup.

```
# service dbus enable
# service avahi_daemon enable
# service netatalk enable
```

In case you're wondering, dbus came in as a dependency during the package installation. It needs to run alongside the other services: avahi for network discovery and netatalk since time machine works only with the Apple Filing Protocol (AFP).

Let's start these services right away to move on to the client side.

```
# service dbus start
# service avahi_daemon start
# service netatalk start
```

Check the output of

```
# sockstat -l
```

for the daemons listening on their respective ports and look into `/var/log/afp.log` for any errors.

On the Mac that should be backed up by the newly created time machine, we need to make sure that Time Machine recognizes this (non-Apple official) network volume. To do that, run the following in Terminal.app:

```
$ sudo defaults write com.apple.systempreferences TMShowUnsupportedNet-
workVolumes 1
```

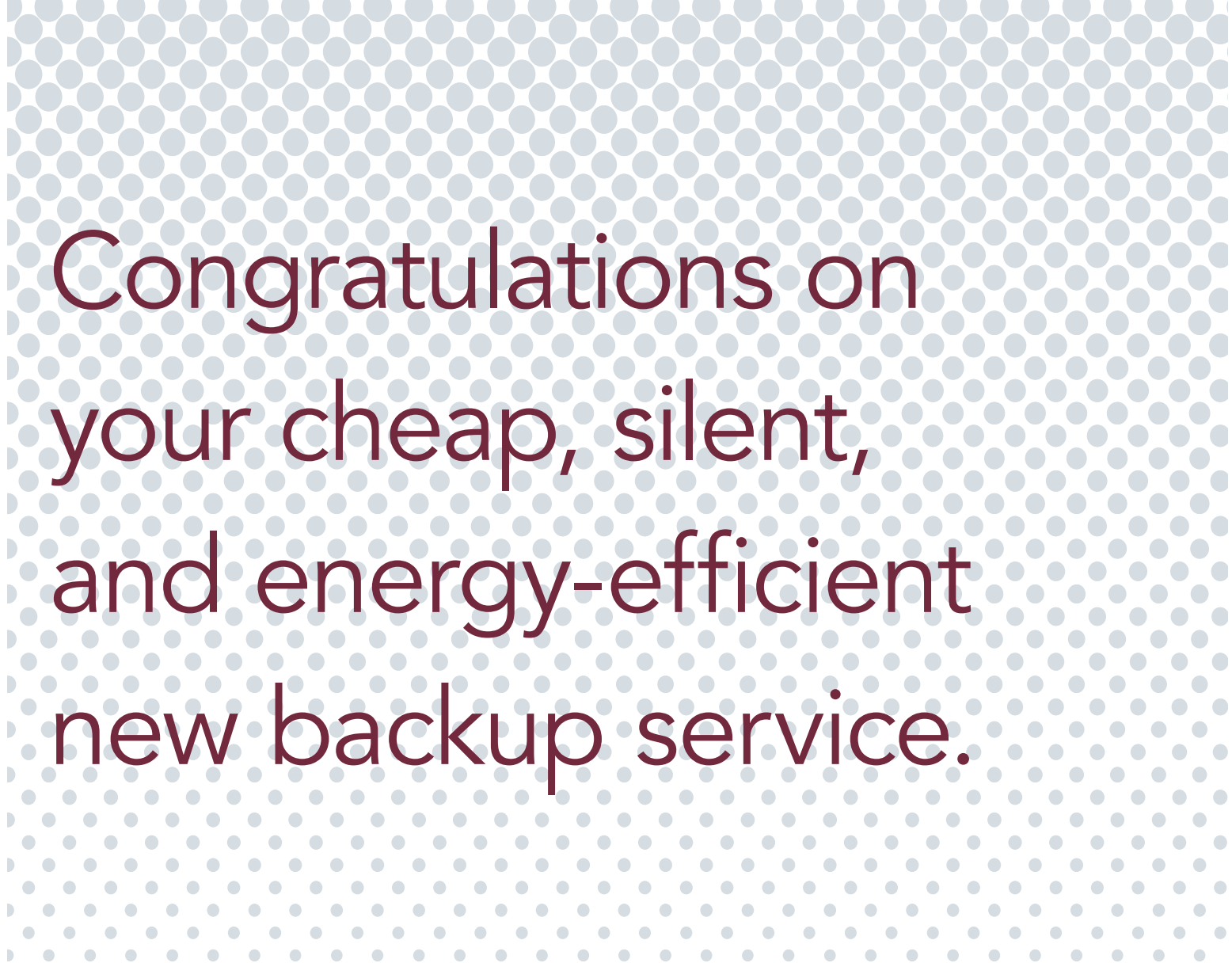
Next, go to the Finder, hit CMD-K (or select “Go to” from the menu and then “Connect to server...”) and enter the following:

```
afp://10.20.30.40
```

Substitute my example with the address or hostname that you configured in `afp.conf` and hit the connect button. Enter your username and password that you have on the time machine server (bcr in my case). If all goes well, the share will be mounted over the network and will appear in your left finder sidebar as an empty drive. When the mapping does not work, check the server’s log file again and make sure you have the proper IP address and user credentials.

Open the system preferences for time machine and click on “Add or remove backup volume”. In there, you should see your mounted share from the server. Select it and for extra protection, check the “encrypt backup” option. This is the only time where you can do this, not afterwards! Yes, you could also use ZFS encryption for the dataset, but I’m fine with this setting for my backup needs.

Time machine will now start creating the initial backup, which will take a long time depending on the number of files on your Mac and their size. Read through the other articles in this Journal to pass the time. Once the initial backup is finished, the service will disconnect automatically and reconnect in regular intervals to copy files that have changed. Test your backup by creating a small text file, wait for it to be backed up, then delete it. Act like you just accidentally deleted an important file and restore it using the Time Machine dialog, letting out a dramatic sigh of relief. That’s all, congratulations on your cheap, silent, and energy-efficient new backup service.



Congratulations on
your cheap, silent,
and energy-efficient
new backup service.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. He serves on the board of directors of the FreeBSD Foundation as vice president. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He’s also teaching a course “Unix for Developers” for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.