Aaron Toponce

{ 2012 12 20 }

ZFS Administration, Part XIII- Sending and Receiving Filesystems

Table of Contents

Now that you're a pro at snapshots, we can move to one of the crown jewels of ZFS- the ability to send and receive full filesystems from one host to another. This is epic, and I am not aware of any other filesystem that can do this without the help of 3rd party tools, such as "dd" and "nc".

ZFS Send

Sending a ZFS filesystem means taking a snapshot of a dataset, and sending the snapshot. This ensures that while sending the data, it will always remain consistent, which is crux for all things ZFS. By default, we send the data to a file. We then can move that single file to an offsite backup, another storage server, or whatever. The advantage a ZFS send has over "dd", is the fact that you do not need to take the filesystem offilne to get at the data. This is a Big Win IMO.

To send a filesystem to a file, you first must make a snapshot of the dataset. After the snapshot has been made, you send the snapshot. This produces an output stream, that must be redirected. As such, you would issue something like the following:

```
# zfs snapshot tank/test@tuesday
# zfs send tank/test@tuesday > /backup/test-tuesday.img
```

Now, your brain should be thinking. You have at your disposal a whole suite of Unix utilities to manipulate data. So, rather than storing the raw data, how about we compress it with the "xz" utility?

```
# zfs send tank/test@tuesday | xz > /backup/test-tuesday.img.xz
```

Want to encrypt the backup? You could use OpenSSL or GnuPG:

```
# zfs send tank/test@tuesday | xz | openssl enc -aes-256-cbc -a -salt > /backup/test-tuesday.img.xz.asc
```

ZFS Receive

Receiving ZFS filesystems is the other side of the coin. Where you have a data stream, you can import that data into a full writable filesystem. It wouldn't make much sense to send the filesystem to an image file, if you can't really do anything with the data in the file.

Just as "zfs send" operates on streams, "zfs receive" does the same. So, suppose we want to receive the "/backup/test-tuesday.img" filesystem. We can receive it into any storage pool, and it will create the necessary dataset.

```
# zfs receive tank/test2 < /backup/test-tuesday.img
```

Of course, in our sending example, I compressed and encrypted a sent filesystem. So, to reverse that process, I do the commands in the reverse order:

```
# openssl enc -d -aes-256-cbc -a -in /storage/temp/testzone.gz.ssl | unxz | zfs receive tank/test2
```

The "zfs recv" command can be used as a shortcut.

Combining Send and Receive

Both "zfs send" and "zfs receive" operate on streams of input and output. So, it would make sense that we can send a filesystem into another. Of course we can do this locally:

```
# zfs send tank/test@tuesday | zfs receive pool/test
```

This is perfectly acceptable, but it doesn't make a lot of sense to keep multiple copies of the filesystem on the same storage server. Instead, it would make better sense to send the filesystem to a remote box. You can do this trivially with OpenSSH:

```
# zfs send tank/test@tuesday | ssh user@server.example.com "zfs receive pool/test"
```

Check out the simplicity of that command. You're taking live, running and consistent data from a snapshot, and sending that data to another box. This is epic for offsite storage backups. On your ZFS storage servers, you would run frequent snapshots of the datasets. Then, as a nightly cron job, you would "zfs send" the latest snapshot to an offsite storage server using "zfs receive". And because you are running a secure, tight ship, you encrypt the data with OpenSSL and XZ. Win.

Conclusion

Again, I can't stress the simplicity of sending and receiving ZFS filesystems. This is one of the biggest features in my book that makes ZFS a serious contender in the storage market. Put it in your nightly cron, and make offsite backups of your data with ZFS sending and receiving. You can send filesystems without unmounting them. You can change dataset properties on the receiving end. All your data remains consistent. You can combine it with other Unix utilities.

It's just pure win.