Configuring a Highly Available Service on FreeBSD – Part 1: HAST

One of the problems a system administrator has when providing services like NFS on a network is that sometimes they are critical to the business and downtime needs to be kept to an absolute minimum. This problem can be solved with tools native to FreeBSD.

What you will learn…

- How to configure HAST
- How to control HAST
- How to recover from HAST failures

What you should know…

- How to install FreeBSD
- How to login to FreeBSD
- How to edit files on FreeBSD

In this series of articles, we will introduce and learn how to use these building blocks to make a service and the underlying storage highly available. As an example we will build a highly available NFS server running on two FreeBSD 9.2 machines called nfs-01 and nfs-02. The underlying principles can be applied to other services as well.

In this first part of the series we will learn how to make storage highly available by using HAST. We will take a look at what HAST is, how to configure it, how to control it and how to recover from failures like a splitbrain situation.

What is HAST?

HAST stands for Highly Available Storage. The main component of HAST is the hastd daemon, which allows the user to transparently store data on two physically separated machines which are connected over TCP/IP. HAST supports both the IPv4 and IPv6 connections. The creation of these connections is always initiated by the primary node. In this active/passive setup, the redundant storage can be accessed only on the active node where a disk-like device is presented under `/dev/hast/<resourcename>`. This `<resourcename>` is a GEOM provider. An important thing to know is that HAST does not configure or change the active (primary) or passive (secondary) role by itself. To automate role switching other tools, like for example CARP, have to be configured to handle the failover.

How to configure HAST

The main configuration of the hastd daemon is done in the `/etc/hast.conf` file. This file can consist of a global section, a node specific section and a resource specific section. Let's explore the basic configuration for our setup as described in Listing 1.

In the global section we see the *timeout 20* line, which sets the default timeout for the connection between the hastd daemon on the nodes. This global timeout could be overridden in the node specific and resource specific sections if we wanted to.

If we look at the node specific section for nfs-01 (see Listing 2), we first note the *on nfs-01 {* line. Which specifies that this part is valid for the machine called nfs-01. One advantage of using this construction is that it is possible to use the same configuration file on all nodes, because a node will only pick up the global parts and the parts for itself. It will thus ignore the on `<othernodename>` parts of the configuration file.

The second line of the node specific section for nfs-01 says:

```
pidfile /var/run/hastd.pid
```

Which indicates that the pidfile used by hastd on nfs-01 should be placed in `/var/run/hastd.pid` (which is the default). The last line closes the node specific section.

Let us now continue by looking at the resource specific section (Listing 3). We first see the *resource* keyword followed by the name of the resource *sharedbynfs.* This means that the resource is called *sharedbynfs* and will become available under `/dev/hast/sharedbynfs` on the primary node when hastd has been started and initiated.

If we look at the node specific part of the resource section, we see two configuration options for every node. First the *local* directive which specifies the local device used on this node to use as a backing device for hast. In this example we will use the `/dev/da1` disk. The second line (*remote*) specifies the name of the other node. So for nfs-01 it is nfs-02 and for nfs-02 it is nfs-01.

Of course there are more options than specified in the configuration above. You can find the description of them in the hast.conf manual page (man `hast.conf`).

Starting hastd and controlling HAST

Now that the configuration is in place, we have to initialise our resource on both nodes with the `hastctl` command (Listing 4).

This initialisation creates the metadata that hast needs to be able to determine which data still needs to be synchronised between the nodes.

Now that we have our metadata initialised we can start using hast. To start hast, we have to add the line

Listing 1. *Our hast.conf*

```
timeout 20

on nfs-01 {
    pidfile /var/run/hastd.pid
}
on nfs-02 {
    pidfile /var/run/hastd.pid
}

resource sharedbynfs {
    on nfs-01 {
         local /dev/da1
         remote nfs-02
    }
    on nfs-02 {
         local /dev/da1
         remote nfs-01
    }
}
```

Listing 2. *Node nfs-01 specific section from hast.conf*

```
on nfs-01 {
    pidfile /var/run/hastd.pid
}
```

Listing 3. *Resource specific section from hast.conf*

```
resource sharedbynfs {
    on nfs-01 {
         local /dev/da1
         remote nfs-02
    }
    on nfs-02 {
         local /dev/da1
         remote nfs-01
    }
}
```

Listing 4. *Initializing our resource*

```
hastctl create sharedbynfs
```

Listing 5. *Starting hastd*

```
nfs-01# echo 'hastd_enable="YES"' >> /etc/rc.conf
nfs-01# service hastd start
nfs-01# hastctl role primary sharedbynfs
nfs-01# hastctl status

nfs-02# echo 'hastd_enable="YES"' >> /etc/rc.conf
nfs-02# service hastd start
nfs-02# hastctl role secondary sharedbynfs
nfs-02# hastctl status
```

Listing 6. *Putting a filesystem on the sharedbynfs hast resource*

```
nfs-01# newfs -U /dev/hast/sharedbynfs
nfs-01# mkdir /export
nfs-01# mount -o noatime /dev/hast/sharedbynfs /export
```

`hastd_enable="YES"` to `/etc/rc.conf`. Then we have to start hastd and set a role on each node (see listing 5 for the commands). In this example we make nfs-01 the primary node and nfs-02 the secondary node. Also note the use of the *hastctl status* command to check the current status of our hast configuration.

Creating a filesystem

Now that we have a working hast setup, it is time to put a filesystem (*newfs)* on it and make that filesystem available under /export on the primary node (nfs-01) with the mount command. The exact commands to do this are described in Listing 6. Please note that we are using the *noatime* mount option to reduce the number of I/O requests, which in turn reduces the number of synchronisation actions that hastd has to execute.

```
nfs-01# newfs -U /dev/hast/sharedbynfs
nfs-01# mkdir /export
nfs-01# mount -o noatime /dev/hast/sharedbynfs /export
```

Failover

Of course it is nice to have a setup like this, but to be able to put it to good use we must know how to do a manual failover. Assuming both nodes are still up and running this is relatively straight forward. We use our example setup with nfs-01 and nfs-02 to move the primary node from nfs-01 to nfs-02. First we umount the filesystem on nfs-01 and mark nfs-01 as secondary. When nfs-01 has become a secondary node, we can make nfs-02 the primary node, check the filesystem and mount the filesystem on nfs-02 (See listing 7 for the exact commands). It is a good practice to always check the filesystem after a failover, but before mounting. The reason for this is that in case of a failover due to a failing node, we can not be sure that every bit of data has been synchronised to the other side. This means that we can not be sure that the filesystem is in a clean and consistent state.

Listing 7. *Failover from nfs-01 to nfs-02*

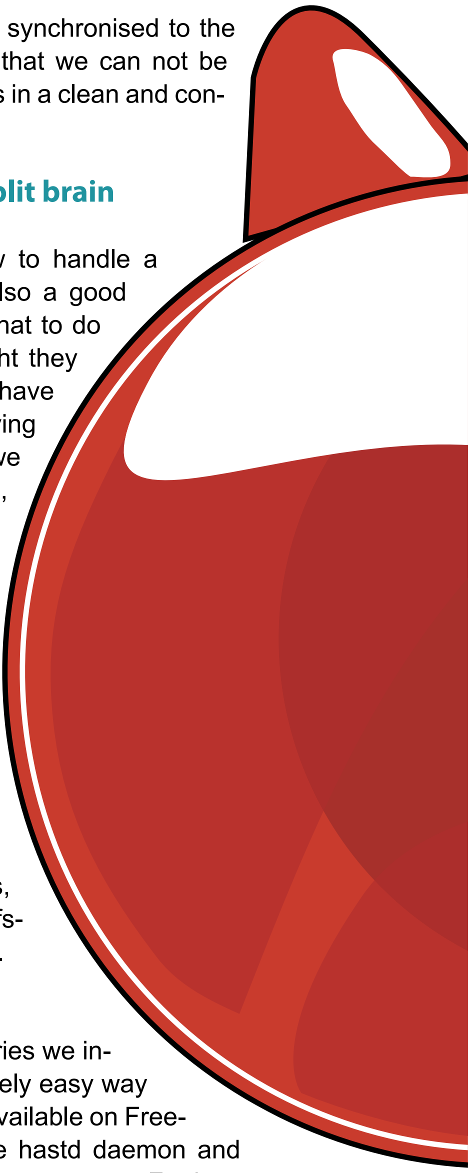
```
nfs-01# umount /export
nfs-01# hastctl role secondary sharedbynfs

nfs-02# hastctl role primary sharedbynfs
nfs-02# fsck -t ufs /dev/hast/sharedbynfs
nfs-02# mount -o noatime /dev/hast/sharedbynfs /export
```

Listing 8. *Recovering from a splitbrain situation*

```
nfs-02# hastctl role init sharedbynfs
nfs-02# hastctl create sharedbynfs
nfs-02# hastctl role secondary sharedbynfs
```

Recovering from a split brain situation

Now that we know how to handle a failover situation, it is also a good idea to take a look at what to do when both nodes thought they were the primary and have written to the underlying storage. In this case we can not avoid data loss, so a decision has to be made which node will resynchronise its data from the other node. That node will have to be disconnected, reinitialised and put in the secondary role after which full data synchronisation will take place. See Listing 8 for the exact commands to do this, where we assume that nfs-02 has to be reinitialised.



Conclusion

In this first part of the series we introduced HAST, a relatively easy way to make storage highly available on FreeBSD. We introduced the hastd daemon and its configuration file `/etc/hast.conf`. Furthermore we learned how to control HAST with the hastctl command and how to recover from a splitbrain situation. Now that we have configured HAST and therefore have created a highly available storage pool for our service, we will learn how to automate failover with CARP and devd in the next part of this series.

JEROEN VAN NIEUWENHUIZEN

Jeroen van Nieuwenhuizen works as a unix consultant for Snow. His free time activities beside playing with FreeBSD include cycling, chess and ice skating.