



Administer BeeGFS Clusters

BeeGFS on NetApp with E-Series Storage

NetApp

March 21, 2024

This PDF was generated from <https://docs.netapp.com/us-en/beegfs/administer-clusters-overview.html> on March 21, 2024. Always check docs.netapp.com for the latest.

Table of Contents

- Administer BeeGFS Clusters 1
 - Overview, key concepts, and terminology 1
 - When to use Ansible versus the pcs tool 2
 - Examine the state of the cluster 2
 - Reconfigure and update 4
 - Service and maintain 8
 - Troubleshoot 14

Administer BeeGFS Clusters

Overview, key concepts, and terminology

Learn how to administer BeeGFS HA clusters after they have been deployed.

Overview

This section is intended for cluster administrators that need to manage BeeGFS HA clusters after they are deployed. Even those familiar with Linux HA clusters should thoroughly read this guide as there are a number of differences in how to manage the cluster, especially around reconfiguration due to the use of Ansible.

Key Concepts

While some of these concepts are introduced on the main [terms and concepts](#) page, it is helpful to reintroduce them in the context of a BeeGFS HA cluster:

Cluster Node: A server running Pacemaker and Corosync services and participating in the HA cluster.

File Node: A cluster node used to run one or more BeeGFS management, metadata, or storage services.

Block Node: A NetApp E-Series storage system that provides block storage to file nodes. These nodes do not participate in the BeeGFS HA cluster as they provide their own standalone HA capabilities. Each node consists of two storage controllers that provide high availability at the block layer.

BeeGFS service: A BeeGFS management, metadata or storage service. Each file node will run one or more services that will use volumes on the block node to store their data.

Building Block: A standardized deployment of BeeGFS file nodes, E-Series block nodes, and the BeeGFS services running on them that simplifies scaling out a BeeGFS HA cluster / file system following a NetApp Verified Architecture. Custom HA clusters are also supported, but often follow a similar building block approach to simplify scaling.

BeeGFS HA Cluster: A scalable number of file nodes used to run BeeGFS services backed by block nodes to store BeeGFS data in a highly available fashion. Built on industry-proven open-source components Pacemaker and Corosync using Ansible for packaging and deployment.

Cluster services: Refers to Pacemaker and Corosync services running on each node participating in the cluster. Note it is possible for a node to not run any BeeGFS services and just participate in the cluster as a "tiebreaker" node in the event there is only a need for two file nodes.

Cluster resources: For each BeeGFS service running in the cluster you will see a BeeGFS monitor resource and a resource group containing resources for BeeGFS target(s), IP address(es) (floating IPs), and the BeeGFS service itself.

Ansible: A tool for software provisioning, configuration management, and application-deployment, enabling infrastructure as code. It is how BeeGFS HA clusters are packaged to simplify the process of deploying, reconfiguring and updating BeeGFS on NetApp.

pcs: A command line interface available from any of the file nodes in the cluster used to query and control the state of nodes and resources in the cluster.

Common Terminology

Failover: Each BeeGFS service has a preferred file node it will run on unless that node fails. When a BeeGFS service is running on non-preferred/secondary file node it is said to be in failover.

Failback: The act of moving BeeGFS services from a non-preferred file node back to their preferred node.

HA pair: Two file nodes that can access the same set of block nodes are sometimes referred to as an HA pair. This is a common term used throughout NetApp to refer to two storage controllers or nodes that can "take over" for each other.

Maintenance Mode: Disables all resource monitoring and prevents Pacemaker from moving or otherwise managing resources in the cluster (also see the section on [maintenance mode](#)).

HA cluster: One or more file nodes running BeeGFS services that can failover between multiple nodes in the cluster to create a highly available BeeGFS file system. Often file nodes are configured into HA pairs that are able to run a subset of the BeeGFS services in the cluster.

When to use Ansible versus the pcs tool

When should you use Ansible versus the pcs command line tool to manage the HA cluster?

All cluster deployment and reconfiguration tasks should be completed using Ansible from an external Ansible control node. Temporary changes in the cluster state (e.g., placing nodes in and out of standby) will typically be performed by logging into one node of the cluster (preferably one that is not degraded or about to undergo maintenance) and using the pcs command line tool.

Modifying any of the cluster configuration including resources, constraints, properties, and the BeeGFS services themselves should always be done using Ansible. Maintaining an up-to-date copy of the Ansible inventory and playbook (ideally in source control to track changes) is part of maintaining the cluster. When you need to make changes to the configuration, update the inventory and rerun the Ansible playbook that imports the BeeGFS HA role.

The HA role will handle placing the cluster into maintenance mode then making any necessary changes before restarting BeeGFS or cluster services to apply the new configuration. As full node reboots aren't typically needed outside the initial deployment, rerunning Ansible is generally considered a "safe" procedure, but always recommended during maintenance windows or off-hours in case any BeeGFS services need to restart. These restarts shouldn't typically cause application errors, but may hurt performance (which some applications may handle better than others).

Rerunning Ansible is also an option when you want to return the entire cluster back to a fully optimal state, and may in some cases be able to recover the state of the cluster more easily than using pcs. Especially during an emergency where the cluster is down for some reason, once all nodes are back up rerunning Ansible may more quickly and reliably recover the cluster than attempting to use pcs.

Examine the state of the cluster

Use pcs to view the state of the cluster.

Overview

Running `pcs status` from any of the cluster nodes is the easiest way to see the overall state of the cluster and the status of each resource (such as BeeGFS services and their dependencies). This section walks through what you will find in the output of the `pcs status` command.

Understanding the output from `pcs status`

Run `pcs status` on any cluster node where the cluster services (Pacemaker and Corosync) are started. The top of the output will show you a summary of the cluster:

```
[root@ictad22h01 ~]# pcs status
Cluster name: hacluster
Cluster Summary:
  * Stack: corosync
  * Current DC: ictad22h01 (version 2.0.5-9.el8_4.3-ba59be7122) -
partition with quorum
  * Last updated: Fri Jul  1 13:37:18 2022
  * Last change:  Fri Jul  1 13:23:34 2022 by root via cibadmin on
ictad22h01
  * 6 nodes configured
  * 235 resource instances configured
```

The section below lists nodes in the cluster:

```
Node List:
  * Node ictad22h06: standby
  * Online: [ ictad22h01 ictad22h02 ictad22h04 ictad22h05 ]
  * OFFLINE: [ ictad22h03 ]
```

This notably indicates any nodes that are in standby or offline. Nodes in standby are still participating in the cluster but marked as ineligible to run resources. Nodes that are offline indicate cluster services are not running on that node, either due to being manually stopped or because the node was rebooted/shutdown.



When nodes first start up, cluster services will be stopped and need to be manually started to avoid accidentally failing back resources to an unhealthy node.

If nodes are in standby or offline due to a non-administrative reason (for example a failure) additional text will be displayed next to the node's state in parenthesis. For example if fencing is disabled and a resource encounters a failure you will see `Node <HOSTNAME>: standby (on-fail)`. Another possible state is `Node <HOSTNAME>: UNCLEAN (offline)`, which will briefly be seen as a node is being fenced, but will persist if fencing failed indicating the cluster cannot confirm the state of the node (this can block the resources from starting on other nodes).

The next section shows a list of all resources in the cluster and their states:

Full List of Resources:

```
* mgmt-monitor      (ocf::eseries:beegfs-monitor):    Started ictad22h01
* Resource Group: mgmt-group:
  * mgmt-FS1        (ocf::eseries:beegfs-target):      Started ictad22h01
  * mgmt-IP1        (ocf::eseries:beegfs-ipaddr2):     Started ictad22h01
  * mgmt-IP2        (ocf::eseries:beegfs-ipaddr2):     Started ictad22h01
  * mgmt-service    (systemd:beegfs-mgmd):            Started ictad22h01
[...]
```

Similar to nodes, additional text will be displayed next to the resource state in parenthesis if there are any issues with the resource. For example if Pacemaker requests a resource stop and it fails to complete within the time allocated, then Pacemaker will attempt to fence the node. If fencing is disabled or the fencing operation fails, the resource state will be `FAILED <HOSTNAME> (blocked)` and Pacemaker will be unable to start it on a different node.

It is worth noting BeeGFS HA clusters make use of a number of BeeGFS optimized custom OCF resource agents. In particular the BeeGFS monitor is responsible for triggering a failover when BeeGFS resources on a particular node are not available.

Reconfigure and update

Reconfigure the HA cluster and BeeGFS

Use Ansible to reconfigure the cluster.

Overview

Generally reconfiguring any aspect of the BeeGFS HA cluster should be done by updating your Ansible inventory and re-running the `ansible-playbook` command. This includes updating alerts, changing the permanent fencing configuration, or adjusting BeeGFS service configuration. These are adjusted using the `group_vars/ha_cluster.yml` file and a full list of options can be found in the [Specify Common File Node Configuration](#) section.

See below for additional details on select configuration options that administrators should be aware of when performing maintenance or servicing the cluster.

How to Disable and Enable Fencing

Fencing is enabled/required by default when setting up the cluster. In some instances it may be desirable to temporarily disable fencing to ensure nodes aren't accidentally shutdown when performing certain maintenance operations (such as upgrading the operating system). While this can be disabled manually, there are tradeoffs administrators should be aware of.

OPTION 1: Disable fencing using Ansible (recommended).

When fencing is disabled using Ansible, the on-fail action of the BeeGFS monitor is changed from "fence" to "standby". This means if the BeeGFS monitor detects a failure it will attempt to place the node in standby and failover all BeeGFS services. Outside active troubleshooting/testing this is typically more desirable than option 2. The disadvantage is if a resource fails to stop on the original node it will be blocked from starting elsewhere (which is why fencing is typically required for production clusters).

1. In your Ansible inventory at `groups_vars/ha_cluster.yml` add the following configuration:

```
beegfs_ha_cluster_crm_config_options:  
  stonith-enabled: False
```

2. Rerun the Ansible playbook to apply the changes to the cluster.

OPTION 2: Disable fencing manually.

In some instances you may want to temporarily disable fencing without rerunning Ansible, perhaps to facilitate troubleshooting or testing of the cluster.



In this configuration if the BeeGFS monitor detects a failure, the cluster will attempt to stop the corresponding resource group. It will NOT trigger a full failover or attempt to restart or move the impacted resource group to another host. To recover, address any issues then run `pcs resource cleanup` or manually place the node in standby.

Steps:

1. To determine if fencing (stonith) is globally enabled or disabled run: `pcs property show stonith-enabled`
2. To disable fencing run: `pcs property set stonith-enabled=false`
3. To enable fencing run: `pcs property set stonith-enabled=true`

Note: This setting will be overridden the next time you run the Ansible playbook.

Update the HA cluster and BeeGFS

Use Ansible to update BeeGFS and the HA cluster.

Overview

BeeGFS is versioned following a `major.minor.patch` versioning scheme and BeeGFS HA Ansible roles are provided for each supported BeeGFS `major.minor` version (for example `beegfs_ha_7_2` and `beegfs_ha_7_3`). Each of the HA roles is pinned to the latest BeeGFS patch version at the time the Ansible collection was released.

Ansible should be used for all BeeGFS upgrades, including moving between major, minor, and patch versions of BeeGFS. To update BeeGFS you will first need to update the BeeGFS Ansible collection, which will also pull in the latest fixes and enhancements to the deployment/management automation and underlying HA cluster. Even after updating to the latest version of the collection, BeeGFS will not be upgraded until `ansible-playbook` is ran with the `-e "beegfs_ha_force_upgrade=true"` set.



For more information on BeeGFS versions see the [BeeGFS Upgrade documentation](#).

Tested Upgrade Paths

Each version of the BeeGFS collection is tested with specific versions of BeeGFS to ensure interoperability between all components. Testing is also performed to ensure upgrades can be performed from the BeeGFS

version(s) supported by the last version of the collection, to those supported in the latest release.

Original Version	Upgrade Version	Multirail	Details
7.2.6	7.3.2	Yes	Upgrading beegfs collection from v3.0.1 to v3.1.0, multirail added
7.2.6	7.2.8	No	Upgrading beegfs collection from v3.0.1 to v3.1.0
7.2.8	7.3.1	Yes	Upgrade using beegfs collection v3.1.0, multirail added
7.3.1	7.3.2	Yes	Upgrade using beegfs collection v3.1.0

BeeGFS Upgrade Steps

The follow sections provide the steps to update the BeeGFS Ansible collection and BeeGFS itself. Pay special attention to any extra step(s) for updating BeeGFS major or minor versions.

Step 1: Upgrade BeeGFS Collection

For collection upgrades with access to [Ansible Galaxy](#), run the following command:

```
ansible-galaxy collection install netapp_eseries.beegfs --upgrade
```

For offline collection upgrades, download the collection from [Ansible Galaxy](#) by clicking on the desired `Install Version`` and then `Download tarball`. Transfer the tarball to your Ansible control node and run the following command.

```
ansible-galaxy collection install netapp_eseries-beegfs-<VERSION>.tar.gz  
--upgrade
```

See [Installing Collections](#) for more information.

Step 2: Update Ansible Inventory

Make any required or desired updates to your cluster's Ansible inventory files. See the [Version Upgrade Notes](#) section below for details about your specific upgrade requirements. See the [Use Custom Architectures](#) section for general information on configuring your BeeGFS HA inventory.

Step 3: Update Ansible Playbook (when updating major or minor versions only)

If you are moving between major or minor versions, in the `playbook.yml` file used to deploy and maintain the cluster, update the name of the `beegfs_ha_<VERSION>` role to reflect the desired version. For example, if you wanted to deploy BeeGFS 7.3 this would be `beegfs_ha_7_3`:


```

- hosts: all
  gather_facts: false
  any_errors_fatal: true
  collections:
    - netapp_eseries.beegfs
  tasks:
    - name: Ensure BeeGFS HA cluster is setup.
      ansible.builtin.import_role: # import_role is required for tag
        availability.
        name: beegfs_ha_7_3

```

For more details on the contents of this playbook file see the [Deploy the BeeGFS HA cluster](#) section.

Step 4: Run the BeeGFS Upgrade

To apply the BeeGFS update:

```

ansible-playbook -i inventory.yml beegfs_ha_playbook.yml -e
"beegfs_ha_force_upgrade=true" --tags beegfs_ha

```

Behind the scenes the BeeGFS HA role will handle:

- Ensure the cluster is in an optimal state with each BeeGFS service located on its preferred node.
- Put the cluster in maintenance mode.
- Update the HA cluster components (if needed).
- Upgrade each file node one at a time as follows:
 - Place it into standby and failover its services to the secondary node.
 - Upgrade BeeGFS packages.
 - Fallback back services.
- Move the cluster out of maintenance mode.

Version Upgrade Notes

Upgrading from BeeGFS version 7.2.6 or 7.3.0

Changes to Connection Based Authentication

BeeGFS versions released after 7.3.1 will no longer allow services to start without either specifying a `connAuthFile` or setting `connDisableAuthentication=true` in the service's configuration file. It is highly recommended to enable connection based authentication security. See [BeeGFS Connection Based Authentication](#) for more information.

By default the `beegfs_ha*` roles will generate and distribute this file, also adding it to the Ansible control node at `<playbook_directory>/files/beegfs/<beegfs_mgmt_ip_address>_connAuthFile`. The `beegfs_client` role will also check for the presence of this file and supply it to the clients if available.



If the `beegfs_client` role was not used to configure clients, this file will need to be manually distributed to each client and the `connAuthFile` configuration in the `beegfs-client.conf` file set to use it. When upgrading from a previous version of BeeGFS where connection based authentication was not enabled, clients will loose access unless connection based authentication is disabled as part of the upgrade by setting `beegfs_ha_conn_auth_enabled: false` in `group_vars/ha_cluster.yml` (not recommended).

For additional details and alternate configuration options see the step to configure connection authentication in the [Specify Common File Node Configuration](#) section.

Service and maintain

Failover and failback services

Moving BeeGFS services between cluster nodes.

Overview

BeeGFS services can failover between nodes in the cluster to ensure clients are able to continue accessing the file system if a node experiences a fault, or you need to perform planned maintenance. This section describes various ways administrators can heal the cluster after recovering from a failure, or manually move services between nodes.

Steps

Failover and Failback

Failover (Planned)

Generally when you need to bring a single file node offline for maintenance you'll want to move (or drain) all BeeGFS services from that node. This can be accomplished by first putting the node in standby:

```
pcs node standby <HOSTNAME>
```

After verifying using `pcs status` all resources have been restarted on the alternate file node, you can shutdown or make other changes to the node as needed.

Failback (after a planned failover)

When you are ready to restore BeeGFS services to the preferred node first run `pcs status` and verify in the "Node List" the status is standby. If the node was rebooted it will show offline until you bring the cluster services online:

```
pcs cluster start <HOSTNAME>
```

Once the node is online bring it out of standby with:

```
pcs cluster node unstandby <HOSTNAME>
```

Lastly relocate all BeeGFS services back to their preferred nodes with:

```
pcs resource relocate run
```

Failback (after an unplanned failover)

If a node experience a hardware or other fault, the HA cluster should automatically react and move its services to a healthy node, providing time for administrators take corrective action. Before proceeding reference the [troubleshooting](#) section to determine the cause of the failover and resolve any outstanding issues. Once the node is powered back on and healthy you can proceed with failback.

When a node boots following an unplanned (or planned) reboot, cluster services are not set to start automatically, so you will first need to bring the node online with:

```
pcs cluster start <HOSTNAME>
```

Next cleanup any resource failures and reset the node's fencing history:

```
pcs resource cleanup node=<HOSTNAME>
pcs stonith history cleanup <HOSTNAME>
```

Verify in `pcs status` the node is online and healthy. By default BeeGFS services will not automatically failback to avoid accidentally moving resources back to an unhealthy node. When you are ready return all resources in the cluster back to their preferred nodes with:

```
pcs resource relocate run
```

Moving individual BeeGFS services to alternate file nodes

Permanently move a BeeGFS service to a new file node

If you want to permanently change the preferred file node for an individual BeeGFS service, adjust the Ansible inventory so the preferred node is listed first and rerun the Ansible playbook.

For example in this sample `inventory.yml` file, `ictad22h01` is the preferred file node to run the BeeGFS management service:

```
mgmt:
  hosts:
    ictad22h01:
    ictad22h02:
```

Reversing the order would cause the management services to be preferred on `ictad22h02`:

```
mgmt:
  hosts:
    ictad22h02:
    ictad22h01:
```

Temporarily move a BeeGFS service to an alternate file node

Generally if a node is undergoing maintenance you will want to use the [failover and failback steps](#failover-and-failback) to move all services away from that node.

If for some reason you do need to move an individual service to a different file node run:

```
pcs resource move <SERVICE>-monitor <HOSTNAME>
```



Do not specify individual resources or the resource group. Always specify the name of the monitor for the BeeGFS service you wish to relocate. For example to move the BeeGFS management service to ictad22h02 run: `pcs resource move mgmt-monitor ictad22h02`. This process can be repeated to move one or more services away from their preferred nodes. Verify using `pcs status` the services were relocated/started on the new node.

To move a BeeGFS service back to its preferred node first clear the temporary resource constraints (repeating this step as needed for multiple services):

```
pcs resource clear <SERVICE>-monitor
```

Then when ready to actually move service(s) back to their preferred node(s) run:

```
pcs resource relocate run
```

Note this command will relocate any services that no longer have temporary resource constraints not located on their preferred nodes.

Place the cluster in maintenance mode

Prevent the HA cluster from accidentally reacting to intended changes in the environment.

Overview

Putting the cluster in maintenance mode disables all resource monitoring and prevents Pacemaker from moving or otherwise managing resources in the cluster. All resources will remain running on their original nodes, regardless if there is a temporary failure condition that would prevent them from being accessible. Scenarios where this is recommended/useful include:

- Network maintenance that may temporarily disrupt connections between file nodes and BeeGFS services.
- Block Node upgrades.
- File Node operating system, kernel, or other package updates.

Generally the only reason to manually put the cluster in maintenance mode is to prevent it from reacting to external changes in the environment. If an individual node in the cluster requires physical repair do not use maintenance mode and simply place that node in standby following the procedure above. Note that rerunning Ansible will automatically put the cluster in maintenance mode facilitating most software maintenance including upgrades and configuration changes.

Steps

To check if the cluster is in maintenance mode run:

```
pcs property show maintenance-mode
```

This will return false when the cluster is operating normally. To enable maintenance mode run:

```
pcs property set maintenance-mode=true
```

You can verify by running `pcs status` and ensuring all resources show "(unmanaged)". To take the cluster out of maintenance mode run:

```
pcs property set maintenance-mode=false
```

Stop and start the cluster

Gracefully stopping and starting the HA cluster.

Overview

This section describes how to gracefully shutdown and restart the BeeGFS cluster. Example scenarios where this may be required include electrical maintenance or migrating between datacenters or racks.

Steps

If for any reason you need to stop the entire BeeGFS cluster and shutdown all services run:

```
pcs cluster stop --all
```

It is also possible to stop the cluster on individual nodes (which will automatically failover services to another node), though it is recommended to first put the node in standby (see the [failover](#) section):

```
pcs cluster stop <HOSTNAME>
```

To start cluster services and resources on all nodes run:

```
pcs cluster start --all
```

Or start services on a specific node with:

```
pcs cluster start <HOSTNAME>
```

At this point run `pcs status` and verify the cluster and BeeGFS services start on all nodes, and services are running on the nodes you expect.



Depending on the size of the cluster it can take sometime (seconds to minutes) for the entire cluster to stop, or show started in `pcs status`. If `pcs cluster <COMMAND>` hangs for more than five minutes, before running "Ctrl+C" to cancel the command, login to each node of the cluster and use `pcs status` to see if cluster services (Corosync/Pacemaker) are still running on that node. From any node where the cluster is still active you can check what resources are blocking the cluster. Manually address the issue and the command should either complete or can be rerun to stop any remaining services.

Replace file nodes

Replacing a file node if the original server is faulty.

Overview

This is an overview of the steps needed to replace a file node in the cluster. These steps presume the file node failed due to a hardware issue, and was replaced with a new identical file node.

Steps:

1. Physically replace the file node and restore all cabling to the block node and storage network.
2. Reinstall the operating system on the file node including adding Red Hat subscriptions.
3. Configure management and BMC networking on the file node.
4. Update the Ansible inventory if the hostname, IP, PCIe-to-logical interface mappings, or anything else changed about the new file node. Generally this is not needed if the node was replaced with identical server hardware and you are using the original network configuration.
 - a. For example if the hostname changed, create (or rename) the node's inventory file (`host_vars/<NEW_NODE>.yaml`) then in the Ansible inventory file (`inventory.yaml`), replace the old node's name with the new node name:

```

all:
  ...
  children:
    ha_cluster:
      children:
        mgmt:
          hosts:
            node_h1_new:  # Replaced "node_h1" with "node_h1_new"
            node_h2:

```

5. From one of the other nodes in the cluster, remove the old node: `pcs cluster node remove <HOSTNAME>`.



DO NOT PROCEED BEFORE RUNNING THIS STEP.

6. On the Ansible control node:
 - a. Remove the old SSH key with:

```
`ssh-keygen -R <HOSTNAME_OR_IP>`
```

- b. Configure passwordless SSH to the replace node with:

```
ssh-copy-id <USER>@<HOSTNAME_OR_IP>
```

7. Rerun the Ansible playbook to configure the node and add it to the cluster:

```
ansible-playbook -i <inventory>.yaml <playbook>.yaml
```

8. At this point, run `pcs status` and verify the replaced node is now listed and running services.

Expand or shrink the cluster

Add or remove building blocks from the cluster.

Overview

This section documents various considerations and options to adjust the size of your BeeGFS HA cluster. Typically cluster size is adjusted by adding or removing building blocks, which are typically two file nodes setup as an HA pair. It is also possible to add or remove individual file nodes (or other types of cluster nodes) if needed.

Adding a Building Block to the Cluster

Considerations

Growing the cluster by adding additional building blocks is a straightforward process. Before you begin keep in mind restrictions around the minimum and maximum number of cluster nodes in each individual HA cluster, and determine if you should add nodes to the existing HA cluster, or create a new HA cluster. Typically each building block consists of two file nodes, but three nodes is the minimum number of nodes per cluster (to establish quorum), and ten is the recommended (tested) maximum. For advanced scenarios it is possible to add a single "tiebreaker" node that does not run any BeeGFS services when deploying a two node cluster. Please contact NetApp support if you are considering such a deployment.

Keep in mind these restrictions and any anticipated future cluster growth when deciding how to expand the cluster. For example if you have a six node cluster and need to add four more nodes, it would be recommended to just start a new HA cluster.



Remember, a single BeeGFS file system can consist of multiple independent HA clusters. This allows file systems to continue scaling far past the recommended/hard limits of the underlying HA cluster components.

Steps

When adding a building block to your cluster, you will need to create the `host_vars` files for each of the new file nodes and block nodes (E-Series arrays). The names of these hosts need to be added to the inventory, along with the new resources that are to be created. The corresponding `group_vars` files will need to be created for each new resource. See the [Use custom architectures](#) section for details.

After creating the correct files, all that is needed is to rerun the automation using the command:

```
ansible-playbook -i <inventory>.yaml <playbook>.yaml
```

Removing a Building Block from the Cluster

There are a number of considerations to keep in mind when you need to retire a building block, for example:

- What BeeGFS services are running in this building block?
- Are just the file nodes retiring and the block nodes should be attached to new file nodes?
- If the entire building block is being retired, should the data be moved to a new building block, dispersed into existing nodes in the cluster, or moved to a new BeeGFS file system or other storage system?
- Can this happen during an outage or should it be done non-disruptively?
- Is the building block actively in use, or does it primarily contain data that is no-longer active?

Because of the diverse possible starting points and desired end states, please contact NetApp support so we can identify and help implement the best strategy based on your environment and requirements.

Troubleshoot

Troubleshooting a BeeGFS HA cluster.

Overview

This section walks through how to investigate and troubleshoot various failures and other scenarios that may arise when operating a BeeGFS HA cluster.

Troubleshooting Guides

Investigating Unexpected Failovers

When a node is unexpectedly fenced and its services moved to another node, the first step should be to see if the cluster indicates any resource failures at the bottom of `pcs status`. Typically nothing will be present if fencing completed successfully and the resources were restarted on another node.

Generally the next step will be to search through the systemd logs using `journalctl` on any one of the remaining file nodes (Pacemaker logs are synchronized on all nodes). If you know the time when the failure occurred you can start the search just before the failure occurred (generally at least ten minutes prior is recommended):

```
journalctl --since "<YYYY-MM-DD HH:MM:SS>"
```

The following sections show common text you can `grep` for in the logs to further narrow down the investigation.

Steps to Investigate/Resolve

Step 1: Check if the BeeGFS monitor detected a failure:

If the failover was triggered by the BeeGFS monitor you should see an error (if not proceed to the next step).

```
journalctl --since "<YYYY-MM-DD HH:MM:SS>" | grep -i unexpected
[...]
Jul 01 15:51:03 ictad22h01 pacemaker-schedulerd[9246]: warning:
Unexpected result (error: BeeGFS service is not active!) was recorded for
monitor of meta_08-monitor on ictad22h02 at Jul 1 15:51:03 2022
```

In this instance BeeGFS service `meta_08` stopped for some reason. To continue troubleshooting we should boot `ictad22h02` and review logs for the service at `/var/log/beegfs-meta-meta_08_tgt_0801.log`. For example, the BeeGFS service could have encountered an application error due to an internal issue or problem with the node.



Unlike the logs from Pacemaker, logs for BeeGFS services are not distributed to all nodes in the cluster. To investigate those types of failures, the logs from the original node where the failure occurred are required.

Possible issues that could be reported by the monitor include:

- Target(s) are not accessible!
 - Description: Indicates the block volumes were not accessible.
 - Troubleshooting:

- If the service also failed to start on the alternate file node, confirm the block node is healthy.
- Check for any physical issues that would prevent access to the block nodes from this file node, for example faulty InfiniBand adapters or cables.
- Network is not reachable!
 - Description: None of the adapters used by clients to connect to this BeeGFS service were online.
 - Troubleshooting:
 - If multiple/all file nodes were impacted, check if there was a fault on the network used to connect the BeeGFS clients and file system.
 - Check for any physical issues that would prevent access to the clients from this file node, for example faulty InfiniBand adapters or cables.
- BeeGFS service is not active!
 - Description: A BeeGFS service stopped unexpectedly.
 - Troubleshooting:
 - On the file node that reported the error, check the logs for the impacted BeeGFS service to see if it reported a crash. If this happened, open a case with NetApp support so the crash can be investigated.
 - If there are no errors reported in the BeeGFS log, check the journal logs to see if systemd logged a reason the service was stopped. In some scenarios the BeeGFS service may not have been given a chance to log any messages before the process was terminated (for example if someone ran `kill -9 <PID>`).

Step 2: Check if the node left the cluster unexpectedly

In the event the node suffered some catastrophic hardware failure (e.g., the system board died) or there was a kernel panic or similar software issue, the BeeGFS monitor will not report an error. Instead look for the hostname and you should see messages from Pacemaker indicating the node was lost unexpectedly:

```
journalctl --since "<YYYY-MM-DD HH:MM:SS>" | grep -i <HOSTNAME>
[...]
Jul 01 16:18:01 ictad22h01 pacemaker-attrd[9245]: notice: Node ictad22h02
state is now lost
Jul 01 16:18:01 ictad22h01 pacemaker-controld[9247]: warning:
Stonith/shutdown of node ictad22h02 was not expected
```

Step 3: Verify Pacemaker was able to fence the node

In all scenarios you should see Pacemaker attempt to fence the node to verify it is actually offline (exact messages may vary by cause of the fencing):

```
Jul 01 16:18:02 ictad22h01 pacemaker-schedulerd[9246]: warning: Cluster
node ictad22h02 will be fenced: peer is no longer part of the cluster
Jul 01 16:18:02 ictad22h01 pacemaker-schedulerd[9246]: warning: Node
ictad22h02 is unclean
Jul 01 16:18:02 ictad22h01 pacemaker-schedulerd[9246]: warning:
Scheduling Node ictad22h02 for STONITH
```

If the fencing action completes successfully you will see messages like:

```
Jul 01 16:18:14 ictad22h01 pacemaker-fenced[9243]: notice: Operation
'off' [2214070] (call 27 from pacemaker-controld.9247) for host
'ictad22h02' with device 'fence_redfish_2' returned: 0 (OK)
Jul 01 16:18:14 ictad22h01 pacemaker-fenced[9243]: notice: Operation
'off' targeting ictad22h02 on ictad22h01 for pacemaker-
controld.9247@ictad22h01.786df3a1: OK
Jul 01 16:18:14 ictad22h01 pacemaker-controld[9247]: notice: Peer
ictad22h02 was terminated (off) by ictad22h01 on behalf of pacemaker-
controld.9247: OK
```

If the fencing action failed for some reason, then the BeeGFS services will be unable to restart on another node to avoid risking data corruption. That would be an issue to investigate separately, if for example the fencing device (PDU or BMC) was inaccessible or misconfigured.

Address Failed Resource Actions (found at the bottom of pcs status)

If a resource required to run a BeeGFS service fails, a failover will be triggered by the BeeGFS monitor. If this occurs there will likely be no "Failed Resource Actions" listed at the bottom of `pcs status` and you should refer to the steps on how to [failback after an unplanned failover](#).

Otherwise there should generally only be two scenarios where you will see "Failed Resource Actions".

Steps to Investigate/Resolve

Scenario 1: A temporary or permanent issue was detected with a fencing agent and it was restarted or moved to another node.

Some fencing agents are more reliable than others, and each will implement their own method of monitoring to ensure the fencing device is ready. In particular the Redfish fencing agent has been seen to report failed resource actions like the following even though it will still show started:

```
* fence_redfish_2_monitor_60000 on ictad22h01 'not running' (7):
call=2248, status='complete', exitreason='', last-rc-change='2022-07-26
08:12:59 -05:00', queued=0ms, exec=0ms
```

A fencing agent reporting failed resource actions on a particular node is not expected to trigger a failover of the BeeGFS services running on that node. It should simply be automatically restarted on the same or a different node.

Steps to resolve:

1. If the fencing agent consistently refuses to run on all or a subset of nodes, check if those nodes are able to connect to the fencing agent, and verify the fencing agent is configured correctly in the Ansible inventory.
 - a. For example if a Redfish (BMC) fencing agent is running on the same node as it is responsible for fencing, and the OS management and BMC IPs are on the same physical interface, some network switch configurations will not allow communication between the two interfaces (to prevent network loops). By default the HA cluster will attempt to avoid placing fencing agents on the node they are responsible for fencing, but this can happen in some scenarios/configurations.
2. Once all issues are resolved (or if the issue appeared to be ephemeral), run `pcs resource cleanup` to reset the failed resource actions.

Scenario 2: The BeeGFS monitor detected an issue and triggered a failover, but for some reason resources could not start on a secondary node.

Provided fencing is enabled and the resource wasn't blocked from stopping on the original node (see the troubleshooting section for "standby (on-fail)"), the most likely reasons include problems starting the resource on a secondary node because:

- The secondary node was already offline.
- A physical or logical configuration issue prevented the secondary from accessing the block volumes used as BeeGFS targets.

Steps to resolve:

1. For each entry in the failed resource actions:
 - a. Confirm the failed resource action was a start operation.
 - b. Based on the resource indicated and the node specified in the failed resource actions:
 - i. Look for and correct any external issues that would prevent the node from starting the specified resource. For example if BeeGFS IP address (floating IP) failed to start, verify at least one of the required interfaces is connected/online and cabled to the right network switch. If a BeeGFS target (block device / E-Series volume) is failed, verify the physical connections to the backend block node(s) are connected as expected, and verify the block nodes are healthy.
 - c. If there are no obvious external issues and you desire a root cause for this incident, it is suggested you open a case with NetApp support to investigate before proceeding as the following steps may make root cause analysis (RCA) challenging/impossible.
2. After resolving any external issues:
 - a. Comment out any non-functional nodes from the Ansible inventory.yml file and rerun the full Ansible playbook to ensure all logical configuration is setup correctly on the secondary node(s).
 - i. Note: Don't forget to uncomment these nodes and rerun the playbook once the nodes are healthy and you are ready to failback.
 - b. Alternatively you can attempt to manually recover the cluster:
 - i. Place any offline nodes back online using: `pcs cluster start <HOSTNAME>`
 - ii. Clear all failed resource actions using: `pcs resource cleanup`
 - iii. Run `pcs status` and verify all services start as expected.
 - iv. If needed run `pcs resource relocate run` to move resources back to their preferred node (if it is available).

Common Issues

BeeGFS services don't failover or fallback when requested

Likely issue: The `pcs resource relocate` run command was executed, but never finished successfully.

How to check: Run `pcs constraint --full` and check for any location constraints with an ID of `pcs-relocate-<RESOURCE>`.

How to resolve: Run `pcs resource relocate clear` then rerun `pcs constraint --full` to verify the extra constraints are removed.

One node in pcs status shows "standby (on-fail)" when fencing is disabled

Likely issue: Pacemaker was unable to successfully confirm all resources were stopped on the node that failed.

How to resolve:

1. Run `pcs status` and check for any resources that aren't "started" or show errors at the bottom of the output and resolve any issues.
2. To bring the node back online run `pcs resource cleanup --node=<HOSTNAME>`.

After an unexpected failover, resources show "started (on-fail)" in pcs status when fencing is enabled

Likely issue: A problem occurred that triggered a failover, but Pacemaker was unable to verify the node was fenced. This could happen because fencing was misconfigured or there was an issue with the fencing agent (example: the PDU was disconnected from the network).

How to resolve:

1. Verify the node is actually powered off.



If the node you specify is not actually off, but running cluster services or resources, data corruption/cluster failure WILL occur.

2. Manually confirm fencing with: `pcs stonith confirm <NODE>`

At this point services should finish failing over and be restarted on another healthy node.

Common Troubleshooting Tasks

Restart individual BeeGFS services

Normally if a BeeGFS service needs to be restarted (say to facilitate a configuration change) this should be done by updating the Ansible inventory and rerunning the playbook. In some scenarios it may be desirable to restart individual services to facilitate faster troubleshooting, for example to change the logging level without needing to wait for the entire playbook to run.



Unless any manual changes are also added to the Ansible inventory, they will be reverted the next time the Ansible playbook runs.

Option 1: Systemd controlled restart

If there is a risk the BeeGFS service won't properly restart with the new configuration, first place the cluster in maintenance mode to prevent the BeeGFS monitor from detecting the service is stopped and triggering an unwanted failover:

```
pcs property set maintenance-mode=true
```

If needed make any changes to the services configuration at `/mnt/<SERVICE_ID>/_config/beegfs-.conf` (example: `/mnt/meta_01_tgt_0101/metadata_config/beegfs-meta.conf`) then use systemd to restart it:

```
systemctl restart beegfs-*@<SERVICE_ID>.service
```

Example: `systemctl restart beegfs-meta@meta_01_tgt_0101.service`

Option 2: Pacemaker controlled restart

If you aren't concerned the new configuration could cause the service to stop unexpectedly (for example simply changing the logging level), or you're in a maintenance window and not concerned about downtime you can simply restart the BeeGFS monitor for the service you want to restart:

```
pcs resource restart <SERVICE>-monitor
```

For example to restart the BeeGFS management service: `pcs resource restart mgmt-monitor`

Copyright information

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b) (FEB 2014).

Trademark information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.