Use custom architectures

BeeGFS on NetApp with E-Series Storage

NetApp
August 29, 2024

Table of Contents

Use custom architectures

Overview and requirements

Use any NetApp E/EF-Series storage systems as BeeGFS block nodes and x86 servers as BeeGFS file nodes when deploying BeeGFS high availability clusters using Ansible.

Definitions for terminology used throughout this section can be found on the terms and concepts page.

Introduction

While NetApp verified architectures provide predefined reference configurations and sizing guidance, some customers and partners may prefer to design custom architectures better suited to particular requirements or hardware preferences. One of the primary benefits of choosing BeeGFS on NetApp is the ability to deploy BeeGFS shared-disk HA clusters using Ansible, simplifying cluster management and improving reliability with NetApp authored HA components. The deployment of custom BeeGFS architectures on NetApp is still done using Ansible, maintaining an appliance-like approach on a flexible range of hardware.

This section outlines the general steps needed to deploy BeeGFS file systems on NetApp hardware and use of Ansible to configure BeeGFS file systems. For details on best practices surrounding the design of BeeGFS file systems and optimized examples please refer to the NetApp verified architectures section.

Deployment Overview

Generally deploying a BeeGFS file system involves the following steps:

- Initial set up:
  - Install/cable hardware.
  - Set up file and block nodes.
  - Set up an Ansible control node.
- Define the BeeGFS file system as an Ansible inventory.
- Run Ansible against file and block nodes to deploy BeeGFS.
  - Optionally to set up clients and mount BeeGFS.

Subsequent sections will cover these steps in more detail.

Ansible handles all software provisioning and configuration tasks including:

- Creating/mapping volumes on block nodes.
- Formatting/tuning volumes on file nodes.
- Installing/configuring software on file nodes.
- Establishing the HA cluster and configuring BeeGFS resources and file system services.

Requirements

Support for BeeGFS in Ansible is released on Ansible Galaxy as a collection of roles and modules that automate the end-to-end deployment and management of BeeGFS HA clusters.

BeeGFS itself is versioned following a <major>.<minor>.<patch> versioning scheme and the collection maintains roles for each supported <major>.<minor> version of BeeGFS, for example BeeGFS 7.2 or BeeGFS 7.3. As updates to the collection are released the patch version in each role will be updated to point at the latest available BeeGFS version for that release branch (example: 7.2.8). Each version of the collection is also tested and supported with specific Linux distributions and versions, currently Red Hat for file nodes, and RedHat and Ubuntu for clients. Running other distributions is not supported, and running other versions (especially other major versions) is not recommended.

Ansible Control Node

This node will contain the inventory and playbooks used to manage BeeGFS. It requires:

- Ansible 6.x (ansible-core 2.13)
- Python 3.6 (or later)
- Python (pip) packages: ipaddr and netaddr

It is also recommend you setup passwordless SSH from the control node to all BeeGFS file nodes and clients.

BeeGFS File Nodes

File nodes must run RedHat 9.3 and have access to the HA repository containing required packages (pacemaker, corosync, fence-agents-all, resource-agents). For example the following command can be executed to enable the appropriate repository on RedHat 9:

```
subscription-manager repo-override repo=rhel-9-for-x86_64-
highavailability-rpms --add=enabled:1
```

BeeGFS Client Nodes

A BeeGFS client Ansible role is available to install the BeeGFS client package and manage BeeGFS mount(s). This role has been tested with RedHat 8.4 and Ubuntu 22.04.

If you are not using Ansible to setup the BeeGFS client and mount BeeGFS, any BeeGFS supported Linux distribution and kernel can be used.

Initial Set Up

Install and Cable Hardware

Steps needed to install and cable hardware used to run BeeGFS on NetApp.

Plan the Installation

Each BeeGFS file system will consist of some number of file nodes running BeeGFS services using backend storage provided by some number of block nodes. The file nodes are configured into one or more high availability clusters to provide fault tolerance for BeeGFS services. Each block node is a already an

active/active HA pair. The minimum number of supported file nodes in each HA cluster is three, and the maximum number of supported file nodes in each cluster is ten. BeeGFS file systems can scale beyond ten node by deploying multiple independent HA clusters that work together to provide a single file system namespace.

Commonly each HA cluster is deployed as a series of "building blocks" where some number of file nodes (x86 servers) are directly connected to some number of block nodes (typically E-Series storage systems). This configuration creates an asymmetrical cluster, where BeeGFS services are only able to run on certain file nodes that have access to the backend block storage used for the BeeGFS targets. The balance of file-to-block nodes in each building block and the storage protocol in use for the direct-connects depend on the requirements of a particular installation.

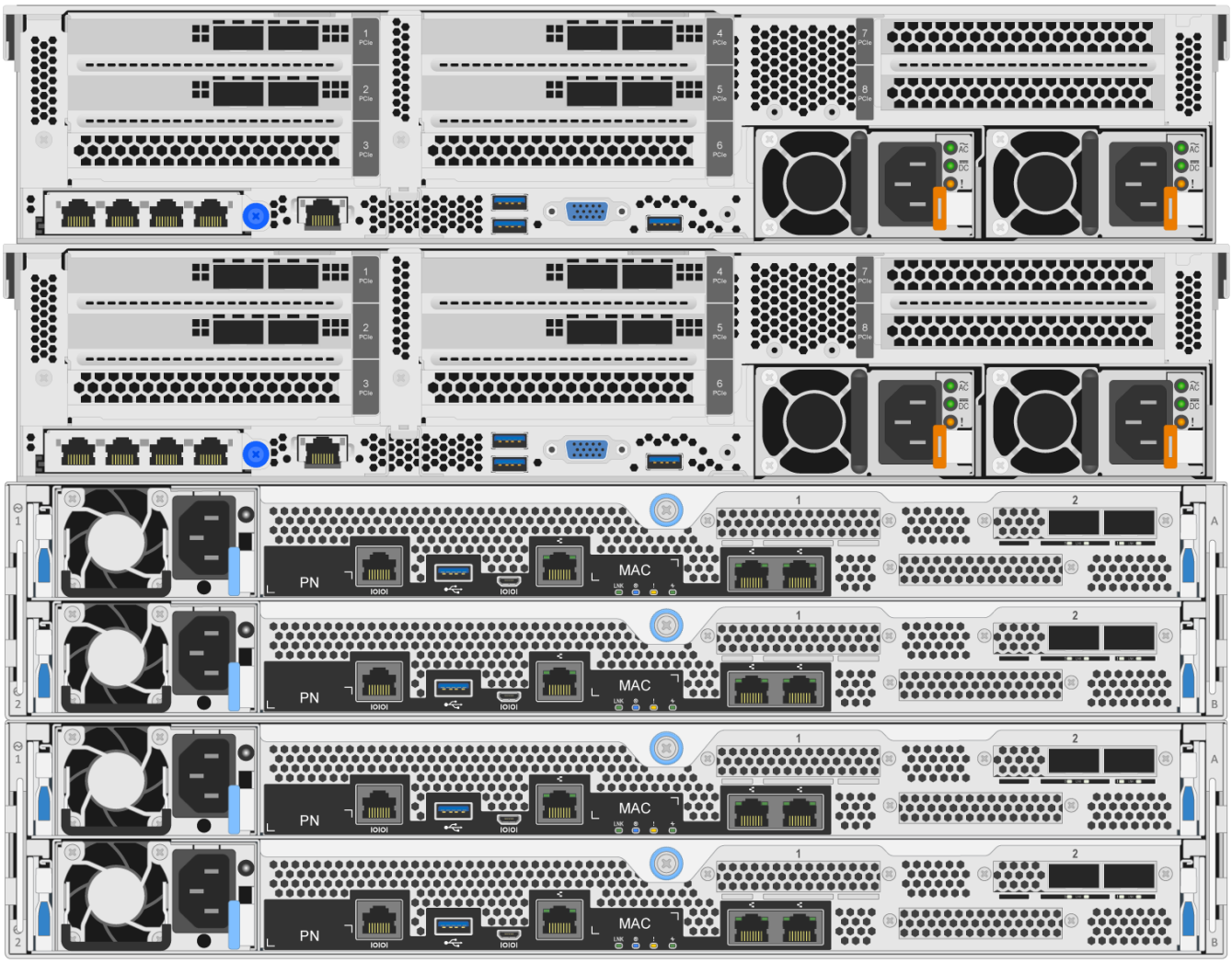
An alternative HA cluster architecture uses a storage fabric (also known as a storage area network or SAN) between the file and block nodes to establish a symmetrical cluster. This allows BeeGFS services to run on any file node in a particular HA cluster. As generally symmetrical clusters are not as cost effective due to the extra SAN hardware, this documentation presumes use of an asymmetrical cluster deployed as a series of one or more building blocks.

Ensure the desired file system architecture for a particular BeeGFS deployment is well understood before proceeding with the installation.

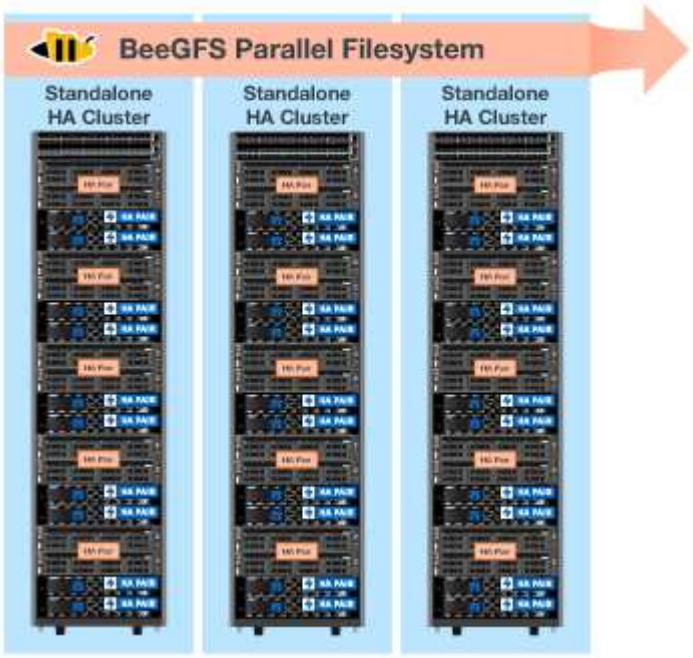
Rack Hardware

When planning the installation it is important all equipment in each building block is racked in adjacent rack units. Best practice is for file nodes to be racked immediately above block nodes in each building block. Follow the documentation for the model(s) of file and block nodes you are using as you install rails and hardware into the rack.

Example of a single building block:

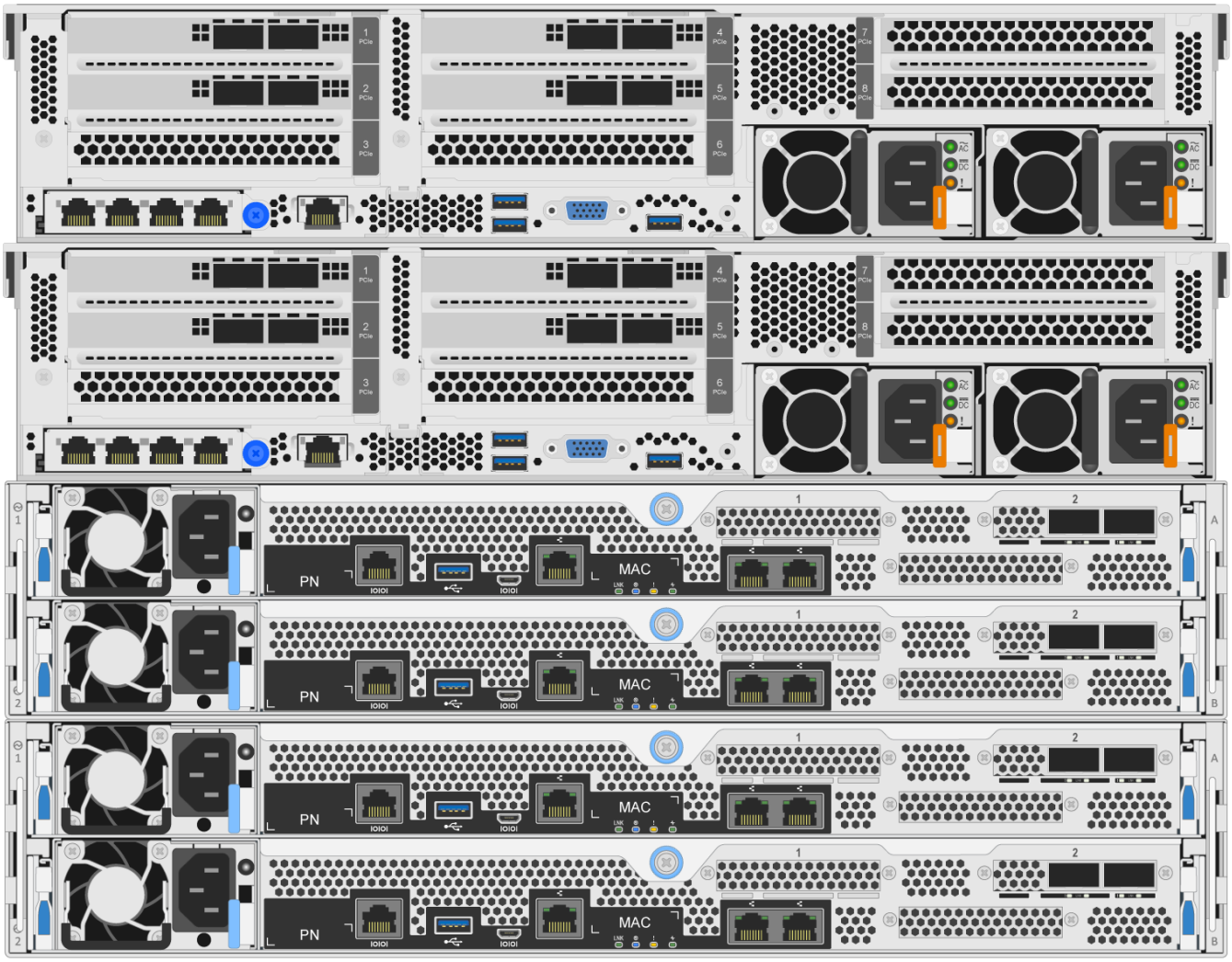


Example of a large BeeGFS installation where there are multiple building blocks in each HA cluster, and multiple HA clusters in the file system:



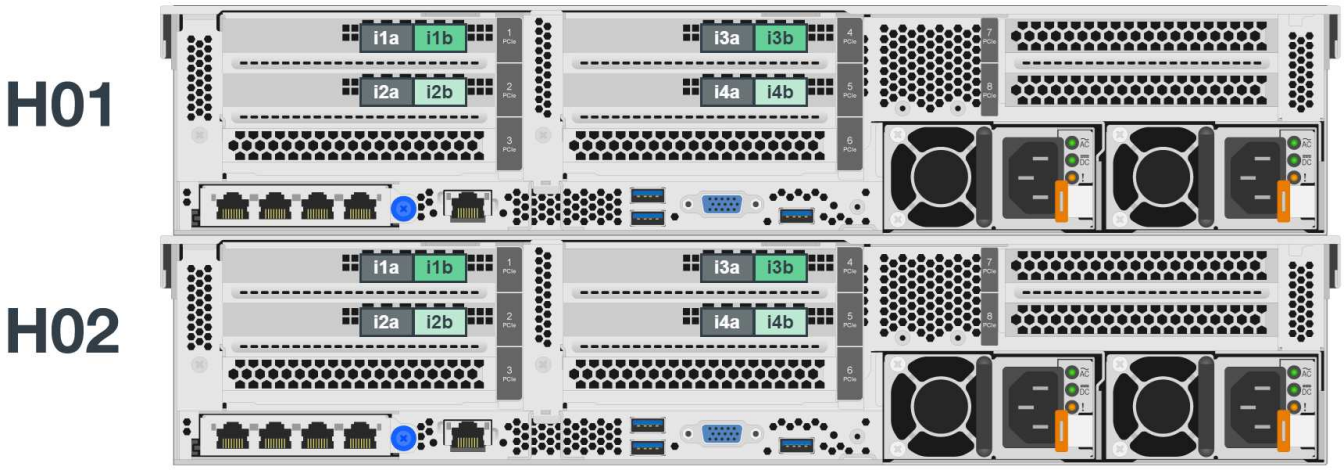
Cable File and Block Nodes

Typically you will direct-connect the HIC ports of the E-Series block nodes to the designated host channel adapter (for InfiniBand protocols) or host bus adapter (for fibre channel and other protocols) ports of the file nodes. The exact way to establish these connections will depend on the desired file system architecture, here is an example based on the second-generation BeeGFS on NetApp verified architecture:



Cable File Nodes to the Client Network

Each file node will have some number of InfiniBand or Ethernet ports designated for BeeGFS client traffic. Depending on the architecture each file node will have one or more connections to a high performance client/storage network, potentially to multiple switches for redundancy and increased bandwidth. Here is an example of client cabling using redundant network switches, where ports highlighted in dark green versus light green are connected to separate switches:



Connect Management Networking and Power

Establish any network connections needed for in-band and out-of-band network.

Connect all power supplies ensuring each file and block node has connections to multiple power distribution units for redundancy (if available).

Set Up File and Block Nodes

Manual steps required to set up file and block nodes before running Ansible.

File Nodes

Configure the Baseboard Management Controller (BMC)

A baseboard management controller (BMC), sometimes referred to as a service processor, is the generic name for the out-of-band management capability built into various server platforms that can provide remote access even if the operating system is not installed or accessible. Vendors typically market this functionality with their own branding. For example, on the Lenovo SR665, the BMC is referred to as the Lenovo XClarity Controller (XCC).

Follow the server vendor's documentation to enable any licenses needed to access this functionality and ensure the BMC is connected to the network and configured appropriately for remote access.

> ⓘ If BMC based fencing using Redfish is desired, ensure Redfish is enabled and the BMC interface is accessible from the OS installed on the file node. Special configuration may be required on the network switch if the BMC and operating share the same physical network interface.

Tune System Settings

Using the system setup (BIOS/UEFI) interface, ensure settings are set to maximize performance. The exact settings and optimal values will vary based on the server model in use. Guidance is provided for verified file node models, otherwise refer to the server vendor's documentation and best practices based on your model.

Install an Operating System

Install a supported operating system based on the file node requirements listed here. Refer to any additional

steps below based on your Linux distribution.

RedHat

Use RedHat Subscription Manager to register and subscribe the system to allow installation of the required packages from the official Red Hat repositories and to limit updates to the supported version of Red Hat: `subscription-manager release --set=<MAJOR_VERSION>.<MINOR_VERSION>`. For instructions, see How to register and subscribe a RHEL system and How to limit updates.

Enable the Red Hat repository containing the packages required for high availability:

```
subscription-manager repo-override --repo=rhel-9-for-x86_64
-highavailability-rpms --add=enabled:1
```

Configure Management Network

Configure any network interfaces needed to allow in-band management of the operating system. The exact steps will depend on the specific Linux distribution and version in use.

Ensure SSH is enabled and all management interfaces are accessible from the Ansible control node.

Update HCA and HBA Firmware

Ensure all HBAs and HCAs are running supported firmware versions listed on the NetApp Interoperability Matrix and upgrade if necessary. Additional recommendations for NVIDIA ConnectX adapters can be found here.

Block Nodes

Follow the steps to get up and running with E-Series to configure the management port on each block node controller and optionally set the storage array name for each system.

No additional configuration beyond ensuring all block nodes are accessible from the Ansible control node is necessary. The remaining system configuration will be applied/maintained using Ansible.

Set Up Ansible Control Node

Set up an Ansible control node to deploy and manage the file system.

Overview

An Ansible control node is a physical or virtual Linux machine used to manage the cluster. It must meet the following requirements:

- Meet the requirements for the BeeGFS HA role including the installed versions of Ansible, Python, and any additional Python packages.
- Meet the official Ansible control node requirements including operating system versions.
- Have SSH and HTTPS access to all file and block nodes.

Detailed installation steps can be found here.

Define the BeeGFS file system

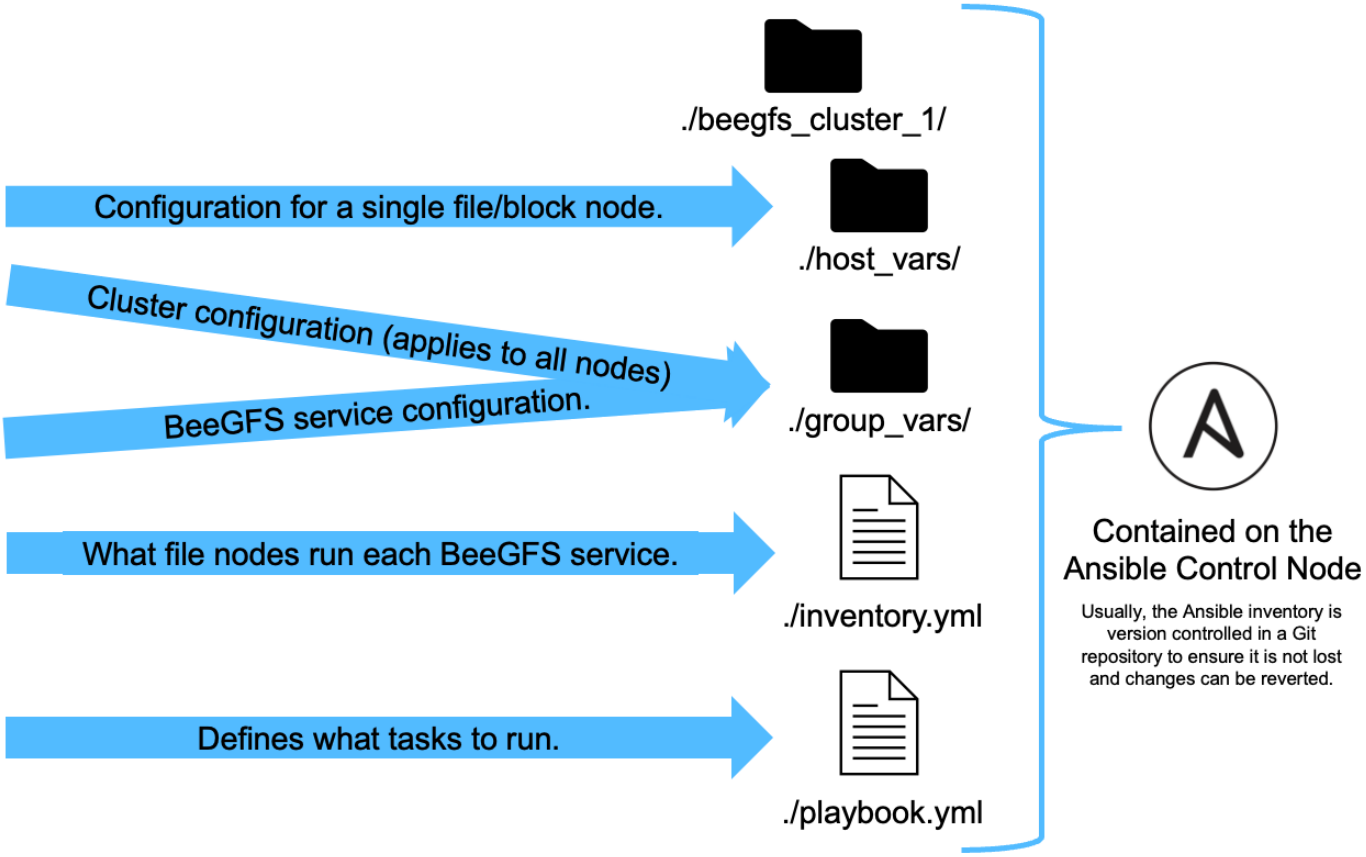
Ansible Inventory Overview

The Ansible inventory is a set of configuration files that define the desired BeeGFS HA cluster.

Overview

It is recommended to follow standard Ansible practices for organizing your inventory, including the use of sub-directories/files instead of storing the entire inventory in one file.

The Ansible inventory for a single BeeGFS HA cluster is organized as follows:



Since a single BeeGFS file system can span multiple HA clusters, it is possible for large installations to have multiple Ansible inventories. Generally it is not recommend to try and define multiple HA clusters as a single Ansible inventory to avoid issues.

Steps

1. On your Ansible control node create an empty directory that will contain the Ansible inventory for the BeeGFS cluster you want to deploy.
   a. If your file system will/may eventually contain multiple HA clusters, it is recommended to first create a directory for the file system, then sub-directories for the inventory representing each HA cluster. For

example:

```
beegfs_file_system_1/
    beegfs_cluster_1/
    beegfs_cluster_2/
    beegfs_cluster_N/
```

2. In the directory containing the inventory for the HA cluster you want to deploy, create two directories `group_vars` and `host_vars` and two files `inventory.yml` and `playbook.yml`.

The following sections walk through defining the contents of each of these files.

Plan the File System

Plan the file system deployment before building out the Ansible inventory.

Overview

Before deploying the file system, you should define what IP addresses, ports, and other configuration will be required by all file nodes, block nodes, and BeeGFS services running in the cluster. While the exact configuration will vary based on the architecture of the cluster, this section defines best practices and steps to follow that are generally applicable.

Steps

1. If you are using an IP based storage protocol (such as iSER, iSCSI, NVMe/IB, or NVMe/RoCE) to connect file nodes to block nodes, fill out the following worksheet for each building block. Each direct connect in a single building block should have a unique subnet, and there should be no overlap with subnets used for client-server connectivity.

File node	IB port	IP address	Block node	IB port	Physical IP	Virtual IP (for EF600 with HDR IB only)
<HOSTNAME>	<PORT>	<IP/SUBNET>	<HOSTNAME>	<PORT>	<IP/SUBNET>	<IP/SUBNET>

> If the file and block nodes in each building block are directly connected you can often reuse the same IPs/scheme for multiple building blocks.

2. Regardless if you are using InfiniBand or RDMA over Converged Ethernet (RoCE) for the storage network, fill out the following worksheet to determine the IP ranges that will be used for HA cluster services, BeeGFS file services, and clients to communicate:

Purpose	InfiniBand port	IP address or range
BeeGFS Cluster IP(s)	<INTERFACE(s)>	<RANGE>
BeeGFS Management	<INTERFACE(s)>	<IP(s)>
BeeGFS Metadata	<INTERFACE(s)>	<RANGE>

Purpose	InfiniBand port	IP address or range
BeeGFS Storage	<INTERFACE(s)>	<RANGE>
BeeGFS Clients	<INTERFACE(s)>	<RANGE>

a. If you are using a single IP subnet only one worksheet is needed, otherwise also fill out a worksheet for the second subnet.

3. Based on the above, for each building block in the cluster, fill out the following worksheet defining what BeeGFS services it will run. For each service specify the preferred/secondary file node(s), network port, floating IP(s), NUMA zone assignment (if required), and what block node(s) will be used for its targets. Refer to the following guidelines when filling out the worksheet:

   a. Specify BeeGFS services as either `mgmt.yml`, `meta_<ID>.yml`, or `storage_<ID>.yml` where ID represents a unique number across all BeeGFS services of that type in this file system. This convention will simplify referring back to this worksheet in subsequent sections while creating files to configure each service.

   b. Ports for BeeGFS services only need to be unique across a particular building block. Ensure services with the same port number cannot ever run on the same file node to avoid port conflicts.

   c. If necessary services can use volumes from more than one block node and/or storage pool (and not all volumes need to be owned by the same controller). Multiple services can also share the same block node and/or storage pool configuration (individual volumes will be defined in a later section).

BeeGFS service (file name)	File Nodes	Port	Floating IPs	NUMA zone	Block node	Storage pool	Owning controller
<SERVICE TYPE>_<ID>.yml	<PREFERRED FILE NODE> <SECONDARY FILE NODE(s)>	<PORT>	<INTERFACE>:<IP/SUBNET> <INTERFACE>:<IP/SUBNET>	<NUMA NODE/ZONE>	<BLOCK NODE>	<STORAGE POOL/VOLUME GROUP>	<A OR B>

For more details on standard conventions, best practices, and filled out example worksheets refer to the best practices and define BeeGFS building blocks sections of the BeeGFS on NetApp Verified Architecture.

Define File and Block Nodes

Configure Individual File Nodes

Specify configuration for individual file nodes using host variables (host_vars).

Overview

This section walks through populating a `host_vars/<FILE_NODE_HOSTNAME>.yml` file for each file node in the cluster. These files should only contain configuration unique to a particular file node. This commonly includes:

- Defining the IP or hostname Ansible should use to connect to the node.
- Configuring additional interfaces and cluster IPs used for HA cluster services (Pacemaker and Corosync) to communicate to other file nodes. By default these services use the same network as the management

interface, but additional interfaces should be available for redundancy. Common practice is to define additional IPs on the storage network, avoiding the need for an additional cluster or management network.

  - The performance of any networks used for cluster communication is not critical for file system performance. With the default cluster configuration generally at least a 1Gb/s network will provide sufficient performance for cluster operations such as synchronizing node states and coordinating cluster resource state changes. Slow/busy networks may cause resource state changes to take longer than usual, and in extreme cases could result in nodes being evicted from the cluster if they cannot send heartbeats in a reasonable time frame.

- Configuring interfaces used for connecting to block nodes over the desired protocol (for example: iSCSI/iSER, NVMe/IB, NVMe/RoCE, FCP, etc.)

Steps

Referencing the IP addressing scheme defined in the Plan the File System section, for each file node in the cluster create a file `host_vars/<FILE_NODE_HOSTNAME>/yml` and populate it as follows:

1. At the top specify the IP or hostname Ansible should use to SSH to the node and manage it:

```
ansible_host: "<MANAGEMENT_IP>"
```

2. Configure additional IPs that can be used for cluster traffic:
   a. If the network type is InfiniBand (using IPoIB):

```
eseries_ipoib_interfaces:
- name: <INTERFACE>  # Example: ib0 or i1b
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

   b. If the network type is RDMA over Converged Ethernet (RoCE):

```
eseries_roce_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

   c. If the network type is Ethernet (TCP only, no RDMA):

```
eseries_ip_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

3. Indicate what IPs should be used for cluster traffic, with preferred IPs listed higher:

```
beegfs_ha_cluster_node_ips:
- <MANAGEMENT_IP> # Including the management IP is typically but not
required.
- <IP_ADDRESS>    # Ex: 100.127.100.1
- <IP_ADDRESS>    # Additional IPs as needed.
```

> IPs configured in step two will not be used as cluster IPs unless they are included in the `beegfs_ha_cluster_node_ips` list. This allows you to configure additional IPs/interfaces using Ansible that can be used for other purposes if desired.

4. If the file node needs to communicate to block nodes over an IP-based protocol, IPs will need to be configured on the appropriate interface, and any packages required for that protocol installed/configured.
   a. If using iSCSI:

```
eseries_iscsi_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
```

   b. If using iSER:

```
eseries_ib_iser_interfaces:
- name: <INTERFACE>  # Example: ib0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
  configure: true # If the file node is directly connected to the
block node set to true to setup OpenSM.
```

   c. If using NVMe/IB:

```
eseries_nvme_ib_interfaces:
- name: <INTERFACE>  # Example: ib0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
  configure: true # If the file node is directly connected to the
block node set to true to setup OpenSM.
```

   d. If using NVMe/RoCE:

```
eseries_nvme_roce_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
```

e. Other Protocols:

   i. If using NVMe/FC, configuring individual interfaces is not required. The BeeGFS cluster deployment will automatically detect the protocol and install/configure requirements as needed. If you are using a fabric to connect file and block nodes, ensure switches are properly zoned following NetApp and your switch vendor's best practices.

   ii. Use of FCP or SAS do not require installing or configuring additional software. If using FCP, ensure switches are properly zoned following NetApp and your switch vendor's best practices.

   iii. Use of IB SRP is not recommended at this time. Use NVMe/IB or iSER depending on what your E-Series block nodes support.

Click here for an example of a complete inventory file representing a single file node.

Advanced: Toggling NVIDIA ConnectX VPI Adapters between Ethernet and InfiniBand Mode

NVIDIA ConnectX-Virtual Protocol Interconnect® (VPI) adapters support both InfiniBand and Ethernet as the transport layer. Switching between modes is not automatically negotiated, and must be configured using the <code>mstconfig</code> tool included in <code>mstflint</code>, an open source package that is part of the NVIDIA Firmare Tools (MFT). Changing the mode of the adapters only need to be done once. This can be done manually, or included in the Ansible inventory as part of any interfaces configured using the <code>eseries-[ib|ib_iser|ipoib|nvme_ib|nvme_roce|roce]_interfaces:</code> section of the inventory, to have it checked/applied automatically.

For example to change an interface current in InfiniBand mode to Ethernet so it can be used for RoCE:

1. For each interface you want to configure specify `mstconfig` as a mapping (or dictionary) that specifies `LINK_TYPE_P<N>` where `<N>` is determined by the HCA's port number for the interface. The `<N>` value can be determined by running `grep PCI_SLOT_NAME /sys/class/net/<INTERFACE_NAME>/device/uevent` and adding 1 to the last number from the PCI slot name and converting to decimal.

   a. For example given `PCI_SLOT_NAME=0000:2f:00.2` (2 + 1 → HCA port 3) → `LINK_TYPE_P3: eth`:

```
eseries_roce_interfaces:
- name: <INTERFACE>
  address: <IP/SUBNET>
  mstconfig:
    LINK_TYPE_P3: eth
```

For additional details refer to the NetApp E-Series Host collection's documentation for the interface type/protocol you are using.

Configure Individual Block Nodes

Specify configuration for individual block nodes using host variables (host_vars).

Overview

This section walks through populating a `host_vars/<BLOCK_NODE_HOSTNAME>.yml` file for each block node in the cluster. These files should only contain configuration unique to a particular block node. This commonly includes:

- The system name (as displayed in System Manager).
- The HTTPS URL for one of the controllers (used to manage the system using its REST API).
- What storage protocol file nodes use to connect to this block node.
- Configuring host interface card (HIC) ports, such as IP addresses (if needed).

Steps

Referencing the IP addressing scheme defined in the Plan the File System section, for each block node in the cluster create a file `host_vars/<BLOCK_NODE_HOSTNAME>/yml` and populate it as follows:

1. At the top specify the system name and the HTTPS URL for one of the controllers:

```
eseries_system_name: <SYSTEM_NAME>
eseries_system_api_url:
https://<MANAGEMENT_HOSTNAME_OR_IP>:8443/devmgr/v2/
```

2. Select the protocol file nodes will use to connect to this block node:
   a. Supported Protocols: `auto`, `iscsi`, `fc`, `sas`, `ib_srp`, `ib_iser`, `nvme_ib`, `nvme_fc`, `nvme_roce`.

```
eseries_initiator_protocol: <PROTOCOL>
```

3. Depending on the protocol in use, the HIC ports may require additional configuration. When needed, HIC port configuration should be defined so the top entry in the configuration for each controller corresponds with with the physical left-most port on each controller, and the bottom port the right-most port. All ports require valid configuration even if they are not currently in use.

> Also see the section below if you are using HDR (200Gb) InfiniBand or 200Gb RoCE with EF600 block nodes.

   a. For iSCSI:

```
eseries_controller_iscsi_port:
  controller_a:         # Ordered list of controller A channel
definition.
    - state:            # Whether the port should be enabled.
Choices: enabled, disabled
      config_method:    # Port configuration method Choices: static,
dhcp
      address:          # Port IPv4 address
      gateway:          # Port IPv4 gateway
      subnet_mask:      # Port IPv4 subnet_mask
      mtu:              # Port IPv4 mtu
    - (...)             # Additional ports as needed.
  controller_b:         # Ordered list of controller B channel
definition.
    - (...)             # Same as controller A but for controller B

# Alternatively the following common port configuration can be
defined for all ports and omitted above:
eseries_controller_iscsi_port_state: enabled         # Generally
specifies whether a controller port definition should be applied
Choices: enabled, disabled
eseries_controller_iscsi_port_config_method: dhcp    # General port
configuration method definition for both controllers. Choices:
static, dhcp
eseries_controller_iscsi_port_gateway:               # General port
IPv4 gateway for both controllers.
eseries_controller_iscsi_port_subnet_mask:           # General port
IPv4 subnet mask for both controllers.
eseries_controller_iscsi_port_mtu: 9000              # General port
maximum transfer units (MTU) for both controllers. Any value greater
than 1500 (bytes).
```

b. For iSER:

```
eseries_controller_ib_iser_port:
  controller_a:    # Ordered list of controller A channel address
definition.
    -              # Port IPv4 address for channel 1
    - (...)        # So on and so forth
  controller_b:    # Ordered list of controller B channel address
definition.
```

c. For NVMe/IB:

```
eseries_controller_nvme_ib_port:
  controller_a:    # Ordered list of controller A channel address
definition.
    -              # Port IPv4 address for channel 1
    - (...)        # So on and so forth
  controller_b:    # Ordered list of controller B channel address
definition.
```

d. For NVMe/RoCE:

```
eseries_controller_nvme_roce_port:
  controller_a:          # Ordered list of controller A channel
definition.
    - state:             # Whether the port should be enabled.
      config_method:     # Port configuration method Choices: static,
dhcp
      address:           # Port IPv4 address
      subnet_mask:       # Port IPv4 subnet_mask
      gateway:           # Port IPv4 gateway
      mtu:               # Port IPv4 mtu
      speed:             # Port IPv4 speed
  controller_b:          # Ordered list of controller B channel
definition.
    - (...)              # Same as controller A but for controller B

# Alternatively the following common port configuration can be
defined for all ports and omitted above:
eseries_controller_nvme_roce_port_state: enabled         # Generally
specifies whether a controller port definition should be applied
Choices: enabled, disabled
eseries_controller_nvme_roce_port_config_method: dhcp    # General
port configuration method definition for both controllers. Choices:
static, dhcp
eseries_controller_nvme_roce_port_gateway:               # General
port IPv4 gateway for both controllers.
eseries_controller_nvme_roce_port_subnet_mask:           # General
port IPv4 subnet mask for both controllers.
eseries_controller_nvme_roce_port_mtu: 4200              # General
port maximum transfer units (MTU). Any value greater than 1500
(bytes).
eseries_controller_nvme_roce_port_speed: auto            # General
interface speed. Value must be a supported speed or auto for
automatically negotiating the speed with the port.
```

e. FC and SAS protocols do not require additional configuration. SRP is not correctly recommended.

For additional options to configure HIC ports and host protocols including the ability to configure iSCSI CHAP refer to the documentation included with the SANtricity collection. Note when deploying BeeGFS the storage pool, volume configuration, and other aspects of provisioning storage will be configured elsewhere, and should not be defined in this file.

Click here for an example of a complete inventory file representing a single block node.

Using HDR (200Gb) InfiniBand or 200Gb RoCE with NetApp EF600 block nodes:

To use HDR (200Gb) InfiniBand with the EF600, a second "virtual" IP must be configured for each physical port. Below is an example of the correct way to configure an EF600 equipped with the dual port InfiniBand HDR HIC:

```
eseries_controller_nvme_ib_port:
  controller_a:
    - 192.168.1.101    # Port 2a (virtual)
    - 192.168.2.101    # Port 2b (virtual)
    - 192.168.1.100    # Port 2a (physical)
    - 192.168.2.100    # Port 2b (physical)
  controller_b:
    - 192.168.3.101    # Port 2a (virtual)
    - 192.168.4.101    # Port 2b (virtual)
    - 192.168.3.100    # Port 2a (physical)
    - 192.168.4.100    # Port 2b (physical)
```

Specify Common File Node Configuration

Specify common file node configuration using group variables (group_vars).

Overview

Configuration that should apple to all file nodes is defined at `group_vars/ha_cluster.yml`. It commonly includes:

- Details on how to connect and login to each file node.
- Common networking configuration.
- Whether automatic reboots are allowed.
- How firewall and selinux states should be configured.
- Cluster configuration including alerting and fencing.
- Performance tuning.
- Common BeeGFS service configuration.

The options set in this file can also be defined on individual file nodes, for example if mixed hardware models are in use, or you have different passwords for each node. Configuration on individual file nodes will take precedence over the configuration in this file.

Steps

Create the file `group_vars/ha_cluster.yml` and populate it as follows:

1. Indicate how the Ansible Control node should authenticate with the remote hosts:

```
ansible_ssh_user: root
ansible_become_password: <PASSWORD>
```

> Particularly for production environments, do not store passwords in plain text. Instead, use Ansible Vault (see Encrypting content with Ansible Vault) or the `--ask-become-pass` option when running the playbook. If the `ansible_ssh_user` is already root, then you can optionally omit the `ansible_become_password`.

2. If you are configuring static IPs on ethernet or InfiniBand interfaces (for example cluster IPs) and multiple interfaces are in the same IP subnet (for example if ib0 is using 192.168.1.10/24 and ib1 is using 192.168.1.11/24), additional IP routing tables and rules must be setup for multi-homed support to work properly. Simply enable the provided network interface configuration hook as follows:

```
eseries_ip_default_hook_templates:
  - 99-multihoming.j2
```

3. When deploying the cluster, depending on the storage protocol it may be necessary for nodes to be rebooted to facilitate discovering remote block devices (E-Series volumes) or apply other aspects of the configuration. By default nodes will prompt before rebooting, but you can allow nodes to restart automatically by specifying the following:

```
eseries_common_allow_host_reboot: true
```

   a. By default after a reboot, to ensure block devices and other services are ready Ansible will wait until the systemd `default.target` is reached before continuing with the deployment. In some scenarios when NVMe/IB is in use, this may not be long enough to initialize, discover, and connect to remote devices. This can result in the automated deployment continuing prematurely and failing. To avoid this when using NVMe/IB also define the following:

```
eseries_common_reboot_test_command: "! systemctl status
eseries_nvme_ib.service || systemctl --state=exited | grep
eseries_nvme_ib.service"
```

4. A number of firewall ports are required for BeeGFS and HA cluster services to communicate. Unless you wish to configure the firwewall manually (not recommended), specify the following to have required firewall zones created and ports opened automatically:

```
beegfs_ha_firewall_configure: True
```

5. At this time SELinux is not supported, and it is is recommended the state be set to disabled to avoid conflicts (especially when RDMA is in use). Set the following to ensure SELinux is disabled:

```
eseries_beegfs_ha_disable_selinux: True
eseries_selinux_state: disabled
```

6. Configure authentication so file nodes are able to communicate, adjusting the defaults as needed based on your organizations policies:

```
beegfs_ha_cluster_name: hacluster                    # BeeGFS HA cluster
name.
beegfs_ha_cluster_username: hacluster                # BeeGFS HA cluster
username.
beegfs_ha_cluster_password: hapassword               # BeeGFS HA cluster
username's password.
beegfs_ha_cluster_password_sha512_salt: randomSalt # BeeGFS HA cluster
username's password salt.
```

7. Based on the Plan the File System section specify the BeeGFS management IP for this file system:

```
beegfs_ha_mgmtd_floating_ip: <IP ADDRESS>
```

> While seemingly redundant, `beegfs_ha_mgmtd_floating_ip` is important when you scale the BeeGFS file system beyond a single HA cluster. Subsequent HA clusters are deployed without an additional BeeGFS management service and point at the management service provided by the first cluster.

8. Enable email alerts if desired:

```
beegfs_ha_enable_alerts: True
# E-mail recipient list for notifications when BeeGFS HA resources
change or fail.
beegfs_ha_alert_email_list: ["<EMAIL>"]
# This dictionary is used to configure postfix service
(/etc/postfix/main.cf) which is required to set email alerts.
beegfs_ha_alert_conf_ha_group_options:
      # This parameter specifies the local internet domain name. This is
optional when the cluster nodes have fully qualified hostnames (i.e.
host.example.com)
      mydomain: <MY_DOMAIN>
beegfs_ha_alert_verbosity: 3
#  1) high-level node activity
#  3) high-level node activity + fencing action information + resources
(filter on X-monitor)
#  5) high-level node activity + fencing action information + resources
```

9. Enabling fencing is strongly recommended, otherwise services can be blocked from starting on secondary nodes when the primary node fails.
   a. Enable fencing globally by specifying the following:

```
beegfs_ha_cluster_crm_config_options:
  stonith-enabled: True
```

      i. Note any supported cluster property can also be specified here if needed. Adjusting these is not typically needed, as the BeeGFS HA role ships with a number of well tested defaults.

   b. Next select and configure a fencing agent:
      i. OPTION 1: To enable fencing using APC Power Distribution Units (PDUs):

```
beegfs_ha_fencing_agents:
  fence_apc:
    - ipaddr: <PDU_IP_ADDRESS>
      login: <PDU_USERNAME>
      passwd: <PDU_PASSWORD>
      pcmk_host_map:
"<HOSTNAME>:<PDU_PORT>,<PDU_PORT>;<HOSTNAME>:<PDU_PORT>,<PDU_PORT>
"
```

      ii. OPTION 2: To enable fencing using the Redfish APIs provided by the Lenovo XCC (and other BMCs):

```
redfish: &redfish
  username: <BMC_USERNAME>
  password: <BMC_PASSWORD>
  ssl_insecure: 1 # If a valid SSL certificate is not available
specify "1".

beegfs_ha_fencing_agents:
  fence_redfish:
    - pcmk_host_list: <HOSTNAME>
      ip: <BMC_IP>
      <<: *redfish
    - pcmk_host_list: <HOSTNAME>
      ip: <BMC_IP>
      <<: *redfish
```

iii. For details on configuring other fencing agents refer to the RedHat Documentation.

10. The BeeGFS HA role can apply many different tuning parameters to help further optimize performance. These include optimizing kernel memory utilization and block device I/O, among other parameters. The role ships with a reasonable set of defaults based on testing with NetApp E-Series block nodes, but by default these aren't applied unless you specify:

```
beegfs_ha_enable_performance_tuning: True
```

a. If needed also specify any changes to the default performance tuning here. See the full performance tuning parameters documentation for additional details.

11. To ensure floating IP addresses (sometimes known as logical interfaces) used for BeeGFS services can fail over between file nodes, all network interfaces must be named consistently. By default network interface names are generated by the kernel, which is not guaranteed to generate consistent names, even across identical server models with network adapters installed in the same PCIe slots. This is also useful when creating inventories before the equipment is deployed and generated interface names are known. To ensure consistent device names, based on a block diagram of the server or `lshw -class network -businfo` output, specify the desired PCIe address-to-logical interface mapping as follows:

a. For InfiniBand (IPoIB) network interfaces:

```
eseries_ipoib_udev_rules:
  "<PCIe ADDRESS>": <NAME> # Ex: 0000:01:00.0: i1a
```

b. For Ethernet network interfaces:

```
eseries_ip_udev_rules:
  "<PCIe ADDRESS>": <NAME> # Ex: 0000:01:00.0: e1a
```

To avoid conflicts when interfaces are renamed (preventing them from being renamed), you should not use any potential default names such as eth0, ens9f0, ib0, or ibs4f0. A common naming convention is to use 'e' or 'i' for Ethernet or InfiniBand, followed by the PCIe slot number, and a letter to indicate the the port. For example the second port of an InfiniBand adapter installed in slot 3 would be: i3b.

If you are using a verified file node model, click here example PCIe address-to-logical port mappings.

12. Optionally specify configuration that should apply to all BeeGFS services in the cluster. Default configuration values can be found here, and per-service configuration is specified elsewhere:
    a. BeeGFS Management service:

```
beegfs_ha_beegfs_mgmtd_conf_ha_group_options:
  <OPTION>: <VALUE>
```

    b. BeeGFS Metadata services:

```
beegfs_ha_beegfs_meta_conf_ha_group_options:
  <OPTION>: <VALUE>
```

    c. BeeGFS Storage services:

```
beegfs_ha_beegfs_storage_conf_ha_group_options:
  <OPTION>: <VALUE>
```

13. As of BeeGFS 7.2.7 and 7.3.1 connection authentication must be configured or explicitly disabled. There are a few ways this can be configured using the Ansible based deployment:
    a. By default the deployment will automatically configure connection authentication, and generate a `connauthfile` that will be distributed to all file nodes and used with the BeeGFS services. This file will also be placed/maintained on the Ansible control node at `<INVENTORY>/files/beegfs/<sysMgmtdHost>_connAuthFile` where it should be maintained (securely) for reuse with clients that need to access this file system.
        i. To generate a new key specify `-e "beegfs_ha_conn_auth_force_new=True` when running the Ansible playbook. Note this is ignored if a `beegfs_ha_conn_auth_secret` is defined.
        ii. For advanced options refer to the full list of defaults included with the BeeGFS HA role.
    b. A custom secret can be used by defining the following in `ha_cluster.yml`:

```
beegfs_ha_conn_auth_secret: <SECRET>
```

    c. Connection authentication can be disabled entirely (NOT recommended):

```
beegfs_ha_conn_auth_enabled: false
```

Click here for an example of a complete inventory file representing common file node configuration.

Using HDR (200Gb) InfiniBand with NetApp EF600 block nodes:

To use HDR (200Gb) InfiniBand with the EF600 the subnet manager must support virtualization. If file and block nodes are connected using a switch, this will need to be enabled on the subnet manager manager for the overall fabric.

If block and file nodes are directly connected using InfiniBand, an instance of `opensm` must be configured on each file node for each interface directly connected to a block node. This is done by specifying `configure: true` when configuring file node storage interfaces.

Currently the inbox version of `opensm` shipped with supported Linux distributions does not support virtualization. Instead it is required you install and configure version of `opensm` from the NVIDIA OpenFabrics Enterprise Distribution (OFED). Although deployment using Ansible is still supported, a few additional steps are required:

1. Using curl or your desired tool, download the packages for the version of OpenSM listed in the technology requirements section from NVIDIA's website to the `<INVENTORY>/packages/` directory. For example:

```
curl -o packages/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
https://linux.mellanox.com/public/repo/mlnx_ofed/23.10-
3.2.2.0/rhel9.3/x86_64/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm

curl -o packages/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
https://linux.mellanox.com/public/repo/mlnx_ofed/23.10-
3.2.2.0/rhel9.3/x86_64/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
```

2. Under `group_vars/ha_cluster.yml` define the following configuration:

```
### OpenSM package and configuration information
eseries_ib_opensm_allow_upgrades: true
eseries_ib_opensm_skip_package_validation: true
eseries_ib_opensm_rhel_packages: []
eseries_ib_opensm_custom_packages:
  install:
    - files:
        add:
          "packages/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm": "/tmp/"
          "packages/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm": "/tmp/"
    - packages:
        add:
          - /tmp/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
          - /tmp/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
  uninstall:
    - packages:
        remove:
          - opensm
          - opensm-libs
      files:
        remove:
          - /tmp/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
          - /tmp/opensm-libs-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm

eseries_ib_opensm_options:
  virt_enabled: "2"
```

Specify Common Block Node Configuration

Specify common block node configuration using group variables (group_vars).

Overview

Configuration that should apple to all block nodes is defined at `group_vars/eseries_storage_systems.yml`. It commonly includes:

- Details on how the Ansible control node should connect to E-Series storage systems used as block nodes.
- What firmware, NVSRAM, and Drive Firmware versions the nodes should run.
- Global configuration including cache settings, host configuration, and settings for how volumes should be

provisioned.

> The options set in this file can also be defined on individual block nodes, for example if mixed hardware models are in use, or you have different passwords for each node. Configuration on individual block nodes will take precedence over the configuration in this file.

Steps

Create the file `group_vars/eseries_storage_systems.yml` and populate it as follows:

1. Ansible does not use SSH to connect to block nodes, and instead uses REST APIs. To achieve this we must set:

```
ansible_connection: local
```

2. Specify the username and password to manage each node. The username can be optionally omitted (and will default to admin), otherwise you can specify any account with admin privileges. Also specify if SSL certificates should be verified, or ignored:

```
eseries_system_username: admin
eseries_system_password: <PASSWORD>
eseries_validate_certs: false
```

> Listing any passwords in plaintext is not recommended. Use Ansible vault or provide the `eseries_system_password` when running Ansible using --extra-vars.

3. Optionally specify what controller firmware, NVSRAM, and drive firmware should be installed on the nodes. These will need to be downloaded to the `packages/` directory before running Ansible. E-Series controller firmware and NVSRAM can be downloaded here and drive firmware here:

```
eseries_firmware_firmware: "packages/<FILENAME>.dlp" # Ex.
"packages/RCB_11.80GA_6000_64cc0ee3.dlp"
eseries_firmware_nvsram: "packages/<FILENAME>.dlp" # Ex.
"packages/N6000-880834-D08.dlp"
eseries_drive_firmware_firmware_list:
  - "packages/<FILENAME>.dlp"
  # Additional firmware versions as needed.
eseries_drive_firmware_upgrade_drives_online: true # Recommended unless
BeeGFS hasn't been deployed yet, as it will disrupt host access if set
to "false".
```

> If this configuration is specified, Ansible will automatically update all firmware including rebooting controllers (if necessary) with with no additional prompts. This is expected to be non-disruptive to BeeGFS/host I/O, but may cause a temporary decrease in performance.

4. Adjust global system configuration defaults. The options and values listed here are commonly recommended for BeeGFS on NetApp, but can be adjusted if needed:

```
eseries_system_cache_block_size: 32768
eseries_system_cache_flush_threshold: 80
eseries_system_default_host_type: linux dm-mp
eseries_system_autoload_balance: disabled
eseries_system_host_connectivity_reporting: disabled
eseries_system_controller_shelf_id: 99 # Required by default.
```

5. Configure global volume provisioning defaults. The options and values listed here are commonly recommended for BeeGFS on NetApp, but can be adjusted if needed:

```
eseries_volume_size_unit: pct # Required by default. This allows volume
capacities to be specified as a percentage, simplifying putting together
the inventory.
eseries_volume_read_cache_enable: true
eseries_volume_read_ahead_enable: false
eseries_volume_write_cache_enable: true
eseries_volume_write_cache_mirror_enable: true
eseries_volume_cache_without_batteries: false
```

6. If needed, adjust the order in which Ansible will select drives for storage pools and volume groups keeping in mind the following best practices:
   a. List any (potentially smaller) drives that should be used for management and/or metadata volumes first, and storage volumes last.
   b. Ensure to balance the drive selection order across available drive channels based on the disk shelf/drive enclosure model(s). For example with the EF600 and no expansions, drives 0-11 are on drive channel 1, and drives 12-23 are on drive channel. Thus a strategy to balance drive selection is to select `disk shelf:drive` 99:0, 99:23, 99:1, 99:22, etc. In the event there is more than one enclosure, the first digit represents the drive shelf ID.

```
# Optimal/recommended order for the EF600 (no expansion):
eseries_storage_pool_usable_drives:
"99:0,99:23,99:1,99:22,99:2,99:21,99:3,99:20,99:4,99:19,99:5,99:18,99
:6,99:17,99:7,99:16,99:8,99:15,99:9,99:14,99:10,99:13,99:11,99:12"
```

Click here for an example of a complete inventory file representing common block node configuration.

Define BeeGFS services

Define the BeeGFS management service

BeeGFS services are configured using group variables (group_vars).

Overview

This section walks through defining the BeeGFS management service. Only one service of this type should exist in the HA cluster(s) for a particular file system. Configuring this service includes defining:

- The service type (management).
- Defining any configuration that should only apply to this BeeGFS service.
- Configuring one or more floating IPs (logical interfaces) where this service can be reached.
- Specifying where/how a volume should be to store data for this service (the BeeGFS management target).

Steps

Create a new file `group_vars/mgmt.yml` and referencing the Plan the File System section populate it as follows:

1. Indicate this file represents the configuration for a BeeGFS management service:

```
beegfs_service: management
```

2. Define any configuration that should apply only to this BeeGFS service. This is not typically required for the management service unless you need to enable quotas, however any supported configuration parameter from `beegfs-mgmtd.conf` can be included. Note the following parameters are configured automatically/elsewhere and should not be specified here: `storeMgmtdDirectory`, `connAuthFile`, `connDisableAuthentication`, `connInterfacesFile`, and `connNetFilterFile`.

```
beegfs_ha_beegfs_mgmtd_conf_resource_group_options:
  <beegfs-mgmt.conf:key>:<beegfs-mgmt.conf:value>
```

3. Configure one or more floating IPs that other services and clients will use to connect to this service (this will automatically set the BeeGFS `connInterfacesFile` option):

```
floating_ips:
  - <INTERFACE>:<IP/SUBNET> # Primary interface. Ex.
i1b:100.127.101.0/16
  - <INTERFACE>:<IP/SUBNET> # Secondary interface(s) as needed.
```

4. Optionally, specify one or more allowed IP subnets which may be used for outgoing communication (this will automatically set the BeeGFS `connNetFilterFile` option):

```
filter_ip_ranges:
  - <SUBNET>/<MASK> # Ex. 192.168.10.0/24
```

5. Specify the BeeGFS management target where this service will store data according to the following guidelines:

a. The same storage pool or volume group name can be used for multiple BeeGFS services/targets, simply ensure to use the same `name`, `raid_level`, `criteria_*`, and `common_*` configuration for each (the volumes listed for each service should be different).

b. Volume sizes should be specified as a percentage of the storage pool/volume group and the total should not exceed 100 across all services/volumes using a particular storage pool/volume group. Note when using SSDs it is recommended to leave some free space in the volume group to maximize SSD performance and wear life (click here for more details).

c. Click here for a full list of configuration options available for the `eseries_storage_pool_configuration`. Note some options such as `state`, `host`, `host_type`, `workload_name`, and `workload_metadata` and volume names are generated automatically and should not be specified here.

```
beegfs_targets:
  <BLOCK_NODE>: # The name of the block node as found in the Ansible
inventory. Ex: netapp_01
    eseries_storage_pool_configuration:
      - name: <NAME> # Ex: beegfs_m1_m2_m5_m6
        raid_level: <LEVEL> # One of: raid1, raid5, raid6, raidDiskPool
        criteria_drive_count: <DRIVE COUNT> # Ex. 4
        common_volume_configuration:
          segment_size_kb: <SEGMENT SIZE> # Ex. 128
        volumes:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
```

Click here for an example of a complete inventory file representing a BeeGFS management service.

Define the BeeGFS metadata service

BeeGFS services are configured using group variables (group_vars).

Overview

This section walks through defining the BeeGFS metadata service. At least one service of this type should exist in the HA cluster(s) for a particular file system. Configuring this service includes defining:

- The service type (metadata).
- Defining any configuration that should only apply to this BeeGFS service.
- Configuring one or more floating IPs (logical interfaces) where this service can be reached.
- Specifying where/how a volume should be to store data for this service (the BeeGFS metadata target).

Steps

Referencing the Plan the File System section, create a file at `group_vars/meta_<ID>.yml` for each metadata service in the cluster, and populate them as follows:

1. Indicate this file represents the configuration for a BeeGFS metadata service:

```
beegfs_service: metadata
```

2. Define any configuration that should apply only to this BeeGFS service. At minimum you must specify the desired TCP and UDP port, however any supported configuration parameter from `beegfs-meta.conf` can also be included. Note the following parameters are configured automatically/elsewhere and should not be specified here: `sysMgmtdHost`, `storeMetaDirectory`, `connAuthFile`, `connDisableAuthentication`, `connInterfacesFile`, and `connNetFilterFile`.

```
beegfs_ha_beegfs_meta_conf_resource_group_options:
  connMetaPortTCP: <TCP PORT>
  connMetaPortUDP: <UDP PORT>
  tuneBindToNumaZone: <NUMA ZONE> # Recommended if using file nodes with
multiple CPU sockets.
```

3. Configure one or more floating IPs that other services and clients will use to connect to this service (this will automatically set the BeeGFS `connInterfacesFile` option):

```
floating_ips:
  - <INTERFACE>:<IP/SUBNET> # Primary interface. Ex.
i1b:100.127.101.1/16
  - <INTERFACE>:<IP/SUBNET> # Secondary interface(s) as needed.
```

4. Optionally, specify one or more allowed IP subnets which may be used for outgoing communication (this will automatically set the BeeGFS `connNetFilterFile` option):

```
filter_ip_ranges:
  - <SUBNET>/<MASK> # Ex. 192.168.10.0/24
```

5. Specify the BeeGFS metadata target where this service will store data according to the following guidelines (this will also automatically configure the `storeMetaDirectory` option):

   a. The same storage pool or volume group name can be used for multiple BeeGFS services/targets, simply ensure to use the same `name`, `raid_level`, `criteria_*`, and `common_*` configuration for each (the volumes listed for each service should be different).

   b. Volume sizes should be specified as a percentage of the storage pool/volume group and the total should not exceed 100 across all services/volumes using a particular storage pool/volume group. Note when using SSDs it is recommended to leave some free space in the volume group to maximize SSD performance and wear life (click here for more details).

   c. Click here for a full list of configuration options available for the `eseries_storage_pool_configuration`. Note some options such as `state`, `host`, `host_type`, `workload_name`, and `workload_metadata` and volume names are generated automatically and should not be specified here.

```
beegfs_targets:
  <BLOCK_NODE>: # The name of the block node as found in the Ansible
inventory. Ex: netapp_01
    eseries_storage_pool_configuration:
      - name: <NAME> # Ex: beegfs_m1_m2_m5_m6
        raid_level: <LEVEL> # One of: raid1, raid5, raid6, raidDiskPool
        criteria_drive_count: <DRIVE COUNT> # Ex. 4
        common_volume_configuration:
          segment_size_kb: <SEGMENT SIZE> # Ex. 128
        volumes:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
```

Click here for an example of a complete inventory file representing a BeeGFS metadata service.

Define the BeeGFS storage service

BeeGFS services are configured using group variables (group_vars).

Overview

This section walks through defining the BeeGFS storage service. At least one service of this type should exist in the HA cluster(s) for a particular file system. Configuring this service includes defining:

- The service type (storage).
- Defining any configuration that should only apply to this BeeGFS service.
- Configuring one or more floating IPs (logical interfaces) where this service can be reached.
- Specifying where/how volume(s) should be to store data for this service (the BeeGFS storage targets).

Steps

Referencing the Plan the File System section, create a file at `group_vars/stor_<ID>.yml` for each storage service in the cluster, and populate them as follows:

1. Indicate this file represents the configuration for a BeeGFS storage service:

   ```
   beegfs_service: storage
   ```

2. Define any configuration that should apply only to this BeeGFS service. At minimum you must specify the desired TCP and UDP port, however any supported configuration parameter from `beegfs-storage.conf` can also be included. Note the following parameters are configured automatically/elsewhere and should not be specified here: `sysMgmtdHost`, `storeStorageDirectory`, `connAuthFile`, `connDisableAuthentication`, `connInterfacesFile`, and `connNetFilterFile`.

```
beegfs_ha_beegfs_storage_conf_resource_group_options:
  connStoragePortTCP: <TCP PORT>
  connStoragePortUDP: <UDP PORT>
  tuneBindToNumaZone: <NUMA ZONE> # Recommended if using file nodes with
multiple CPU sockets.
```

3. Configure one or more floating IPs that other services and clients will use to connect to this service (this will automatically set the BeeGFS `connInterfacesFile` option):

```
floating_ips:
  - <INTERFACE>:<IP/SUBNET> # Primary interface. Ex.
i1b:100.127.101.1/16
  - <INTERFACE>:<IP/SUBNET> # Secondary interface(s) as needed.
```

4. Optionally, specify one or more allowed IP subnets which may be used for outgoing communication (this will automatically set the BeeGFS `connNetFilterFile` option):

```
filter_ip_ranges:
  - <SUBNET>/<MASK> # Ex. 192.168.10.0/24
```

5. Specify the BeeGFS storage target(s) where this service will store data according to the following guidelines (this will also automatically configure the `storeStorageDirectory` option):
   a. The same storage pool or volume group name can be used for multiple BeeGFS services/targets, simply ensure to use the same `name`, `raid_level`, `criteria_*`, and `common_*` configuration for each (the volumes listed for each service should be different).
   b. Volume sizes should be specified as a percentage of the storage pool/volume group and the total should not exceed 100 across all services/volumes using a particular storage pool/volume group. Note when using SSDs it is recommended to leave some free space in the volume group to maximize SSD performance and wear life (click here for more details).
   c. Click here for a full list of configuration options available for the `eseries_storage_pool_configuration`. Note some options such as `state`, `host`, `host_type`, `workload_name`, and `workload_metadata` and volume names are generated automatically and should not be specified here.

```
beegfs_targets:
  <BLOCK_NODE>: # The name of the block node as found in the Ansible
inventory. Ex: netapp_01
    eseries_storage_pool_configuration:
      - name: <NAME> # Ex: beegfs_s1_s2
        raid_level: <LEVEL> # One of: raid1, raid5, raid6,
raidDiskPool
        criteria_drive_count: <DRIVE COUNT> # Ex. 4
        common_volume_configuration:
          segment_size_kb: <SEGMENT SIZE> # Ex. 128
        volumes:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
        # Multiple storage targets are supported / typical:
          - size: <PERCENT> # Percent of the pool or volume group to
allocate to this volume. Ex. 1
            owning_controller: <CONTROLLER> # One of: A, B
```

Click here for an example of a complete inventory file representing a BeeGFS storage service.

Map BeeGFS services to file nodes

Specify what file nodes can run each BeeGFS service using the `inventory.yml` file.

Overview

This section walks through how to create the `inventory.yml` file. This includes listing all block nodes and specifying what file nodes can run each BeeGFS service.

Steps

Create the file `inventory.yml` and populate it as follows:

1. From the top of the file, create the standard Ansible inventory structure:

```
# BeeGFS HA (High Availability) cluster inventory.
all:
  children:
```

2. Create a group containing all block nodes participating in this HA cluster:

```
# Ansible group representing all block nodes:
eseries_storage_systems:
  hosts:
    <BLOCK NODE HOSTNAME>:
    <BLOCK NODE HOSTNAME>:
    # Additional block nodes as needed.
```

3. Create a group that will contain all BeeGFS services in the cluster, and the file nodes that will run them:

```
# Ansible group representing all file nodes:
ha_cluster:
  children:
```

4. For each BeeGFS service in the cluster, define the preferred and any secondary file node(s) that should run that service:

```
    <SERVICE>: # Ex. "mgmt", "meta_01", or "stor_01".
      hosts:
        <FILE NODE HOSTNAME>:
        <FILE NODE HOSTNAME>:
        # Additional file nodes as needed.
```

Click here for an example of a complete inventory file.

Deploy the BeeGFS file system

Ansible Playbook Overview

Deploying and managing BeeGFS HA clusters using Ansible.

Overview

The previous sections walked through the steps needed to build an Ansible inventory representing a BeeGFS HA cluster. This section introduces the Ansible automation developed by NetApp to deploy and manage the cluster.

Ansible: Key Concepts

Before proceeding, it is helpful to be familiar with a few key Ansible concepts:

- Tasks to execute against an Ansible inventory are defined in what is known as a **playbook**.
  - Most tasks in Ansible are designed to be **idempotent**, meaning they can be run multiple times to verify the desired configuration/state is still applied without breaking things or making unnecessary updates.
- The smallest unit of execution in Ansible is a **module**.

    - Typical playbooks use multiple modules.
        - Examples: Download a package, update a config file, start/enable a service.
    - NetApp distributes modules to automate NetApp E-Series systems.
- Complex automation is better packaged as a role.
    - Essentially a standard format to distribute a reusable playbook.
    - NetApp distributes roles for Linux hosts and BeeGFS file systems.

BeeGFS HA role for Ansible: Key Concepts

All the automation needed to deploy and manage each version of BeeGFS on NetApp is packaged as an Ansible role and distributed as part of the NetApp E-Series Ansible Collection for BeeGFS:

- This role can be thought of as somewhere between an **installer** and modern **deployment/management** engine for BeeGFS.
    - Applies modern infrastructure as code practices and philosophies to simplify managing storage infrastructure at any scale.
    - Similar to how the Kubespray project allows users to deploy/maintain an entire Kubernetes distribution for scale out compute infrastructure.
- This role is the **software-defined** format NetApp uses to package, distribute, and maintain BeeGFS on NetApp solutions.
    - Strive to create an “appliance-like” experience without needing to distribute an entire Linux distribution or large image.
    - Includes NetApp authored Open Cluster Framework (OCF) compliant cluster resource agents for custom BeeGFS targets, IP addresses, and monitoring that provide intelligent Pacemaker/BeeGFS integration.
- This role is not simply deployment "automation" and is intended to manage the entire file system lifecycle including:
    - Applying per-service or cluster-wide configuration changes and updates.
    - Automating cluster healing and recovery after hardware issues are resolved.
    - Simplifying performance tuning with default values set based on extensive testing with BeeGFS and NetApp volumes.
    - Verifying and correcting configuration drift.

NetApp also provides an Ansible role for BeeGFS clients, that can optionally be used to install BeeGFS and mount file systems to compute/GPU/login nodes.

Deploy the BeeGFS HA cluster

Specify what tasks should run to deploy the BeeGFS HA cluster using a playbook.

Overview

This section covers how to assemble the standard playbook used to deploy/manage BeeGFS on NetApp.

Steps

Create the Ansible Playbook

Create the file `playbook.yml` and populate it as follows:

1. First define a set of tasks (commonly referred to as a play) that should only run on NetApp E-Series block nodes. We use a pause task to prompt before running the installation (to avoid accidental playbook runs), then import the `nar_santricity_management` role. This role handles applying any general system configuration defined in `group_vars/eseries_storage_systems.yml` or individual `host_vars/<BLOCK NODE>.yml` files.

```
- hosts: eseries_storage_systems
  gather_facts: false
  collections:
    - netapp_eseries.santricity
  tasks:
    - name: Verify before proceeding.
      pause:
        prompt: "Are you ready to proceed with running the BeeGFS HA
role? Depending on the size of the deployment and network performance
between the Ansible control node and BeeGFS file and block nodes this
can take awhile (10+ minutes) to complete."
    - name: Configure NetApp E-Series block nodes.
      import_role:
        name: nar_santricity_management
```

2. Define the play that will run against all file and block nodes:

```
- hosts: all
  any_errors_fatal: true
  gather_facts: false
  collections:
    - netapp_eseries.beegfs
```

3. Within this play we can optionally define a set of "pre-tasks" that should run before deploying the HA cluster. This can be useful to verify/install any prerequisites like Python. We can also inject any pre-flight checks, for example verifying the provided Ansible tags are supported:

```
  pre_tasks:
    - name: Ensure a supported version of Python is available on all
file nodes.
      block:
        - name: Check if python is installed.
          failed_when: false
          changed_when: false
          raw: python --version
          register: python_version
```

```
        - name: Check if python3 is installed.
          raw: python3 --version
          failed_when: false
          changed_when: false
          register: python3_version
          when: 'python_version["rc"] != 0 or (python_version["stdout"]
| regex_replace("Python ", "")) is not version("3.0", ">=")'

        - name: Install python3 if needed.
          raw: |
            id=$(grep "^ID=" /etc/*release* | cut -d= -f 2 | tr -d '"')
            case $id in
              ubuntu) sudo apt install python3 ;;
              rhel|centos) sudo yum -y install python3 ;;
              sles) sudo zypper install python3 ;;
            esac
          args:
            executable: /bin/bash
          register: python3_install
          when: python_version['rc'] != 0 and python3_version['rc'] != 0
          become: true

        - name: Create a symbolic link to python from python3.
          raw: ln -s /usr/bin/python3 /usr/bin/python
          become: true
          when: python_version['rc'] != 0
      when: inventory_hostname not in
groups[beegfs_ha_ansible_storage_group]

    - name: Verify any provided tags are supported.
      fail:
        msg: "{{ item }} tag is not a supported BeeGFS HA tag. Rerun
your playbook command with --list-tags to see all valid playbook tags."
      when: 'item not in ["all", "storage", "beegfs_ha",
"beegfs_ha_package", "beegfs_ha_configure",
"beegfs_ha_configure_resource", "beegfs_ha_performance_tuning",
"beegfs_ha_backup", "beegfs_ha_client"]'
      loop: "{{ ansible_run_tags }}"
```

4. Lastly, this play imports the BeeGFS HA role for the version of BeeGFS you want to deploy:

```
tasks:
  - name: Verify the BeeGFS HA cluster is properly deployed.
    import_role:
      name: beegfs_ha_7_4 # Alternatively specify: beegfs_ha_7_3.
```

A BeeGFS HA role is maintained for each supported major.minor version of BeeGFS. This allows users to choose when they want to upgrade major/minor versions. Currently either BeeGFS 7.3.x (`beegfs_7_3`) or BeeGFS 7.2.x (`beegfs_7_2`) are supported. By default both roles will deploy the latest BeeGFS patch version at the time of release, though users can opt to override this and deploy the latest patch if desired. Refer to the latest upgrade guide for more details.

5. Optional: If you wish to define additional tasks, keep in mind if the tasks should be directed to `all` hosts (including the E-Series storage systems) or only the file nodes. If needed define a new play specifically targeting file nodes using `- hosts: ha_cluster`.

Click here for an example of a complete playbook file.

Install the NetApp Ansible Collections

The BeeGFS collection for Ansible and all dependencies are maintained on Ansible Galaxy. On your Ansible control node run the following command to install the latest version:

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

Though not typically recommended, it is also possible to install a specific version of the collection:

```
ansible-galaxy collection install netapp_eseries.beegfs:
==<MAJOR>.<MINOR>.<PATCH>
```

Run the Playbook

From the directory on your Ansible control node containing the `inventory.yml` and `playbook.yml` files, run the playbook as follows:

```
ansible-playbook -i inventory.yml playbook.yml
```

Based on the size of the cluster the initial deployment can take 20+ minutes. If the deployment fails for any reason, simply correct any issues (e.g., miscabling, node wasn't started, etc.), and restart the Ansible playbook.

When specifying common file node configuration, if you choose the default option to have Ansible automatically manage connection based authentication, a `connAuthFile` used as a shared secret can now be found at `<playbook_dir>/files/beegfs/<sysMgmtdHost>_connAuthFile` (by default). Any clients needing to access the file system will need to use this shared secret. This is handled automatically if clients are configured using the BeeGFS client role.

Deploy BeeGFS clients

Optionally, Ansible can be used to configure BeeGFS clients and mount the file system.

Overview

Accessing BeeGFS file systems requires installing and configuring the BeeGFS client on each node that needs to mount the file system. This section documents how to perform these tasks using the available Ansible role.

Steps

Create the Client Inventory File

1. If needed, set up passwordless SSH from the Ansible control node to each of the hosts you want to configure as BeeGFS clients:

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

2. Under `host_vars/`, create a file for each BeeGFS client named `<HOSTNAME>.yml` with the following content, filling in the placeholder text with the correct information for your environment:

```
# BeeGFS Client
ansible_host: <MANAGEMENT_IP>
```

3. Optionally include one of the following if you want to use the NetApp E-Series Host Collection's roles to configure InfiniBand or Ethernet interfaces for clients to connect to BeeGFS file nodes:

   a. If the network type is InfiniBand (using IPoIB):

```
eseries_ipoib_interfaces:
- name: <INTERFACE>  # Example: ib0 or i1b
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

   b. If the network type is RDMA over Converged Ethernet (RoCE):

```
eseries_roce_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

   c. If the network type is Ethernet (TCP only, no RDMA):

```
eseries_ip_interfaces:
- name: <INTERFACE>  # Example: eth0.
  address: <IP/SUBNET> # Example: 100.127.100.1/16
- name: <INTERFACE>  # Additional interfaces as needed.
  address: <IP/SUBNET>
```

4. Create a new file `client_inventory.yml` and specify the user Ansible should use to connect to each client, and the password Ansible should use for privilege escalation (this requires `ansible_ssh_user` be root, or have sudo privileges):

```
# BeeGFS client inventory.
all:
  vars:
    ansible_ssh_user: <USER>
    ansible_become_password: <PASSWORD>
```

Do not store passwords in plain text. Instead, use the Ansible Vault (see the Ansible documentation for Encrypting content with Ansible Vault) or use the `--ask-become-pass` option when running the playbook.

5. In the `client_inventory.yml` file, list all hosts that should be configured as BeeGFS clients under the `beegfs_clients` group, and then refer to the inline comments and uncomment any additional configuration required to build the BeeGFS client kernel module on your system:

```
children:
    # Ansible group representing all BeeGFS clients:
    beegfs_clients:
      hosts:
        <CLIENT HOSTNAME>:
        # Additional clients as needed.

      vars:
        # OPTION 1: If you're using the NVIDIA OFED drivers and they are
already installed:
        #eseries_ib_skip: True # Skip installing inbox drivers when
using the IPoIB role.
        #beegfs_client_ofed_enable: True
        #beegfs_client_ofed_include_path:
"/usr/src/ofa_kernel/default/include"

        # OPTION 2: If you're using inbox IB/RDMA drivers and they are
already installed:
        #eseries_ib_skip: True # Skip installing inbox drivers when
using the IPoIB role.

        # OPTION 3: If you want to use inbox IB/RDMA drivers and need
them installed/configured.
        #eseries_ib_skip: False # Default value.
        #beegfs_client_ofed_enable: False # Default value.
```

When using the NVIDIA OFED drivers, make sure that beegfs_client_ofed_include_path points to the correct "header include path" for your Linux installation. For more information, see the BeeGFS documentation for RDMA support.

6. In the `client_inventory.yml` file, list the BeeGFS file systems you want mounted under any previously defined `vars`:

```
        beegfs_client_mounts:
          - sysMgmtdHost: <IP ADDRESS>  # Primary IP of the BeeGFS
management service.
            mount_point: /mnt/beegfs    # Path to mount BeeGFS on the
client.
            connInterfaces:
              - <INTERFACE> # Example: ibs4f1
              - <INTERFACE>
            beegfs_client_config:
              # Maximum number of simultaneous connections to the same
node.
              connMaxInternodeNum: 128 # BeeGFS Client Default: 12
              # Allocates the number of buffers for transferring IO.
              connRDMABufNum: 36 # BeeGFS Client Default: 70
              # Size of each allocated RDMA buffer
              connRDMABufSize: 65536 # BeeGFS Client Default: 8192
              # Required when using the BeeGFS client with the shared-
disk HA solution.
              # This does require BeeGFS targets be mounted in the
default "sync" mode.
              # See the documentation included with the BeeGFS client
role for full details.
              sysSessionChecksEnabled: false
        # Specify additional file system mounts for this or other file
systems.
```

7. As of BeeGFS 7.2.7 and 7.3.1 connection authentication must be configured or explicitly disabled. Depending how you choose to configure connection based authentication when specifying common file node configuration, you may need to adjust your client configuration:
   a. By default the HA cluster deployment will automatically configure connection authentication, and generate a `connauthfile` that will be placed/maintained on the Ansible control node at `<INVENTORY>/files/beegfs/<sysMgmtdHost>_connAuthFile`. By default the BeeGFS client role is setup to read/distribute this file to the clients defined in `client_inventory.yml`, and no additional action is needed.
      i. For advanced options refer to the full list of defaults included with the BeeGFS client role.
   b. If you choose to specify a custom secret with `beegfs_ha_conn_auth_secret` specify it in the `client_inventory.yml` file as well:

      ```
      beegfs_ha_conn_auth_secret: <SECRET>
      ```

   c. If you choose to disable connection based authentication entirely with `beegfs_ha_conn_auth_enabled`, specify that in the `client_inventory.yml` file as well:

```
beegfs_ha_conn_auth_enabled: false
```

For a full list of supported parameters and additional details refer to the full BeeGFS client documentation. For a complete example of a client inventory click here.

Create the BeeGFS Client Playbook File

1. Create a new file `client_playbook.yml`

```
# BeeGFS client playbook.
- hosts: beegfs_clients
  any_errors_fatal: true
  gather_facts: true
  collections:
    - netapp_eseries.beegfs
    - netapp_eseries.host
  tasks:
```

2. Optional: If you want to use the NetApp E-Series Host Collection's roles to configure interfaces for clients to connect to BeeGFS file systems, import the role corresponding with the interface type you are configuring:
   a. If you are using are using InfiniBand (IPoIB):

```
    - name: Ensure IPoIB is configured
      import_role:
        name: ipoib
```

   b. If you are using are using RDMA over Converged Ethernet (RoCE):

```
    - name: Ensure IPoIB is configured
      import_role:
        name: roce
```

   c. If you are using are using Ethernet (TCP only, no RDMA):

```
    - name: Ensure IPoIB is configured
      import_role:
        name: ip
```

3. Lastly import the BeeGFS client role to install the client software and setup the file system mounts:

```
    # REQUIRED: Install the BeeGFS client and mount the BeeGFS file
system.
    - name: Verify the BeeGFS clients are configured.
      import_role:
        name: beegfs_client
```

For a complete example of a client playbook click here.

Run the BeeGFS Client Playbook

To install/build the client and mount BeeGFS, run the following command:

```
ansible-playbook -i client_inventory.yml client_playbook.yml
```

Verify the BeeGFS deployment

Verify the file system deployment before placing the system in production.

Overview

Before you place the BeeGFS file system in production, perform a few verification checks.

Steps

1. Login to any client and run the following to ensure all expected nodes are present/reachable, and there are no inconsistencies or other issues reported:

```
beegfs-fsck --checkfs
```

2. Shutdown the entire cluster, then restart it. From any file node run the following:

```
pcs cluster stop --all # Stop the cluster on all file nodes.
pcs cluster start --all  # Start the cluster on all file nodes.
pcs status # Verify all nodes and services are started and no failures
are reported (the command may need to be reran a few times to allow time
for all services to start).
```

3. Place each node in standby and verify BeeGFS services are able to failover to secondary node(s). To do this login to any of the file nodes and run the following:

```
pcs status # Verify the cluster is healthy at the start.
pcs node standby <FILE NODE HOSTNAME> # Place the node under test in
standby.
pcs status # Verify services are started on a secondary node and no
failures are reported.
pcs node unstandby <FILE NODE HOSTNAME> # Take the node under test out
of standby.
pcs status # Verify the file node is back online and no failures are
reported.
pcs resource relocate run # Move all services back to their preferred
nodes.
pcs status # Verify services have moved back to the preferred node.
```

4. Use performance benchmarking tools such as IOR and MDTest to verify the file system performance meets expectations. Examples of common tests and parameters used with BeeGFS can be found in the Design Verification section of the BeeGFS on NetApp Verified Architecture.

Additional tests should be performed based on the acceptance criteria defined for a particular site/installation.

Copyright information

Trademark information