Define File and Block Nodes

BeeGFS on NetApp with E-Series Storage

NetApp
January 27, 2026

Table of Contents

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&reg; (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 <a href="https://docs.nvidia.com/networking/display/mftv4270/mft+supported+configurations+and+parameters" target="_blank">NVIDIA Firmare Tools (MFT)</a>. 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 Red Hat 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-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/rhel9.4/x86_64/opensm-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm
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/rhel9.4/x86_64/opensm-libs-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-5.17.2.MLNX20240610.dc7c2998-
0.1.2310322.x86_64.rpm": "/tmp/"
          "packages/opensm-libs-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.