# Red Hat Hyperconverged Infrastructure for Virtualization 1.8

## Replacing failed hosts

How to replace a failed host and restore data and configuration

Last Updated: 2022-02-10

# Red Hat Hyperconverged Infrastructure for Virtualization 1.8 Replacing failed hosts

How to replace a failed host and restore data and configuration

## Legal Notice

## Abstract

This document explains how to replace a failed host and restore data and configuration in a Red Hat Hyperconverged Infrastructure for Virtualization cluster.

# Table of Contents

# MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

# CHAPTER 1. OVERVIEW

In the event of a hardware issue or some other serious problem, you may need to replace a failed host.

Minimize restore time by backing up disk configuration: Chapter 2, *Backing up important files* . This is not required to restore brick contents, but it does make the process faster.

The process differs depending on the following factors:

- Is data on the disks intact?
- Have you backed up or can you safely back up host configuration details?
- Is the failed host a primary volfile server (the server whose FQDN is used to mount gluster volumes, usually the first host in the cluster)?

Follow the process that best fits your situation:

- If your disks are intact, you can reinstall the same host and use the same FQDN regardless of whether the server is the primary volfile server.
  - If you can back up configuration details, follow Chapter 3, *Reusing bricks and restoring configuration from backups*.
  - If you cannot back up configuration details, follow Chapter 4, *Reusing bricks and reconstructing existing brick configuration*.
- If your disks are not intact or you want the replacement host to use a different FQDN:
  - If the host that failed was the primary volfile server, use Chapter 6, *Replacing a primary host using new bricks*.

    **IMPORTANT**

    This process requires that virtual machines are taken offline.
  - If the host that failed was not the primary volfile server, use Chapter 7, *Replacing a non-primary host using new bricks*.

# CHAPTER 2. BACKING UP IMPORTANT FILES

Backing up important configuration files, inventory files, and modified playbooks makes it easy to restore or redeploy your cluster.

Red Hat recommends backing up your configuration after initial deployment, and after confirming the success of any major changes in your cluster. You can also take backups after a node has failed if necessary.

## Prerequisites

- Example playbooks and inventory files are stored in the **/etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment** directory. If you have manually created or modified inventory and playbook files and you are not storing them in this directory, ensure that you know the path to their location.

## Procedure

1. Log in to a hyperconverged host as the root user.
2. Change into the **hc-ansible-deployment** directory and back up the default **archive_config_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp archive_config_inventory.yml archive_config_inventory.yml.bk
   ```

3. Edit the **archive_config_inventory.yml** file with details of the cluster you want to back up.

   **hosts**

   The backend FQDN of each host in the cluster that you want to back up.

   **backup_dir**

   The directory in which to store backup files.

   **nbde_setup**

   If you use Network-Bound Disk Encryption, set this to **true**. Otherwise, set to **false**.

   **upgrade**

   Set to **false**.

   For example:

   ```
   all:
     hosts:
       host1-backend.example.com:
       host2-backend.example.com:
       host3-backend.example.com:
     vars:
       backup_dir: /rhhi-backup
       nbde_setup: true
       upgrade: false
   ```

4. Run the **archive_config.yml** playbook using your updated inventory file with the **backupfiles** tag.

```
# ansible-playbook -i archive_config_inventory.yml archive_config.yml --tags=backupfiles
```

This creates an archive in the **/root** directory specific to each host FQDN in the **hosts** section of the inventory, for example, **/root/rhvh-node-host1-backend.example.com-backup.tar.gz**.

5. Transfer the backup archives to a different machine.

```
# scp /root/rhvh-node-host1-backend.example.com-backup.tar.gz backup-
host.example.com:/backups/
```

# PART I. USING THE SAME HOST FQDN

# CHAPTER 3. REUSING BRICKS AND RESTORING CONFIGURATION FROM BACKUPS

## 3.1. HOST REPLACEMENT PREREQUISITES

- Determine which node to use as the Ansible controller node (the node from which all Ansible playbooks are executed). Red Hat recommends using a healthy node in the same cluster as the failed node as the Ansible controller node.
- If possible, locate a recent backup or create a new backup of the important files (disk configuration or inventory files). See Backing up important files for details.
- Stop brick processes and unmount file systems on the failed host, to avoid file system inconsistency issues.

  ```
  # pkill glusterfsd
  # umount /gluster_bricks/{engine,vmstore,data}
  ```

- Check which operating system is running on your hyperconverged hosts by running the following command:

  ```
  $ nodectl info
  ```

- Reinstall the same operating system on the failed hyperconverged host.

## 3.2. PREPARING THE CLUSTER FOR HOST REPLACEMENT

1. **Verify host state in the Administrator Portal.**
   a. Log in to the Red Hat Virtualization Administrator Portal.
      The host is listed as **NonResponsive** in the Administrator Portal. Virtual machines that previously ran on this host are in the **Unknown** state.
   b. Click **Compute** → **Hosts** and click the Action menu ( ⋮ ).
   c. Click **Confirm host has been rebooted** and confirm the operation.
   d. Verify that the virtual machines are now listed with a state of **Down**.
2. **Update the SSH fingerprint for the failed node.**
   a. Log in to the Ansible controller node as the root user.
   b. Remove the existing SSH fingerprint for the failed node.

      ```
      # sed -i `/failed-host-frontend.example.com/d` /root/.ssh/known_hosts
      # sed -i `/failed-host-backend.example.com/d` /root/.ssh/known_hosts
      ```

   c. Copy the public key from the Ansible controller node to the freshly installed node.

      ```
      # ssh-copy-id root@new-host-backend.example.com
      # ssh-copy-id root@new-host-frontend.example.com
      ```

d. Verify that you can log in to all hosts in the cluster, including the Ansible controller node, using key-based SSH authentication without a password. Test access using all network addresses. The following example assumes that the Ansible controller node is **host1**.

```
# ssh root@host1-backend.example.com
# ssh root@host1-frontend.example.com
# ssh root@host2-backend.example.com
# ssh root@host2-frontend.example.com
# ssh root@new-host-backend.example.com
# ssh root@new-host-frontend.example.com
```

Use **ssh-copy-id** to copy the public key to any host you cannot log into without a password using this method.

```
# ssh-copy-id root@host-frontend.example.com
# ssh-copy-id root@host-backend.example.com
```

## 3.3. RESTORING DISK CONFIGURATION FROM BACKUPS

### Prerequisites

- This procedure assumes you have already performed the backup process in Chapter 2, *Backing up important files* and know the location of your backup files and the address of the backup host.

### Procedure

1. **If the new host does not have multipath configuration, blacklist the devices.**

   a. Create an inventory file for the new host that defines the devices to blacklist.

   ```
   hc_nodes:
    hosts:
      new-host-backend-fqdn.example.com:
        blacklist_mpath_devices:
          - sda
          - sdb
          - sdc
          - sdd
   ```

   b. Run the **gluster_deployment.yml** playbook on this inventory file using the **blacklistdevices** tag.

   ```
   # ansible-playbook -i blacklist-inventory.yml
   /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-
   deployment/tasks/gluster_deployment.yml --tags=blacklistdevices
   ```

2. **Copy backed up configuration details to the new host.**

```
# mkdir /rhhi-backup
# scp backup-host.example.com:/backups/rhvh-node-host1-backend.example.com-
backup.tar.gz /rhhi-backup
# tar -xvf /rhhi-backup/rhvh-node-host1-backend.example.com-backup.tar.gz -C /rhhi-backup
```

3. **Create an inventory file for host restoration.**

   a. Change into the **hc-ansible-deployment** directory and back up the default **archive_config_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp archive_config_inventory.yml archive_config_inventory.yml.bk
   ```

   b. Edit the **archive_config_inventory.yml** file with details of the cluster you want to back up.

   **hosts**

   The backend FQDN of the host that you want to restore (this host).

   **backup_dir**

   The directory in which to store extracted backup files.

   **nbde_setup**

   If you use Network-Bound Disk Encryption, set this to **true**. Otherwise, set to **false**.

   **upgrade**

   Set to **false**.

   For example:

   ```
   all:
     hosts:
       host1-backend.example.com:
     vars:
       backup_dir: /rhhi-backup
       nbde_setup: true
       upgrade: false
   ```

4. **Execute the archive_config.yml playbook.**
   Run the **archive_config.yml** playbook using your updated inventory file with the **restorefiles** tag.

   ```
   # ansible-playbook -i archive_config_inventory.yml archive_config.yml --tags=restorefiles
   ```

5. **(Optional) Configure Network-Bound Disk Encryption (NBDE) on the root disk.**

   a. Create an inventory file for the new host that defines devices to encrypt.

   ```
   hc_nodes:
     hosts:
       new-node-frontend-fqdn.example.com:
         blacklist_mpath_devices:
           - sda
           - sdb
           - sdc
         rootpassphrase: stronGpa55
         rootdevice: /dev/sda2
         networkinterface: eth1
   vars:
     ip_version: IPv4
     ip_config_method: dhcp
   ```

```
gluster_infra_tangservers:
  - url: http://tang-server.example.com:80
```

See Understanding the luks_tang_inventory.yml file for more information about these parameters.

b. Run the **luks_tang_setup.yml** playbook using your inventory file and the **bindtang** tag.

```
# ansible-playbook -i inventory.yml /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/tasks/luks_tang_setup.yml --tags=bindtang
```

## 3.4. CREATING THE NODE_REPLACE_INVENTORY.YML FILE

Define your cluster hosts by creating a **node_replacement_inventory.yml** file.

### Procedure

1. Back up the **node_replace_inventory.yml** file.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
# cp node_replace_inventory.yml node_replace_inventory.yml.bk
```

2. Edit the **node_replace_inventory.yml** file to define your cluster.
   See Appendix C, *Understanding the **node_replace_inventory.yml** file* for more information about this inventory file and its parameters.

## 3.5. EXECUTING THE REPLACE_NODE.YML PLAYBOOK FILE

The **replace_node.yml** playbook reconfigures a Red Hat Hyperconverged Infrastructure for Virtualization cluster to use a new node after an existing cluster node has failed.

### Procedure

1. Execute the playbook.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/
# ansible-playbook -i node_replace_inventory.yml tasks/replace_node.yml --tags=restorepeer
```

## 3.6. FINALIZING HOST REPLACEMENT

After you have replaced a failed host with a new host, follow these steps to ensure that the cluster is connected to the new host and properly activated.

### Procedure

1. **Activate the host.**

   a. Log in to the Red Hat Virtualization Administrator Portal.

   b. Click **Compute → Hosts** and observe that the replacement host is listed with a state of **Maintenance**.

c. Select the host and click **Management → Activate**.

d. Wait for the host to reach the **Up** state.

2. **Attach the gluster network to the host.**

    a. Click **Compute → Hosts** and select the host.

    b. Click **Network Interfaces → Setup Host Networks**.

    c. Drag and drop the newly created network to the correct interface.

    d. Ensure that the **Verify connectivity between Host and Engine** checkbox is checked.

    e. Ensure that the **Save network configuration** checkbox is checked.

    f. Click **OK** to save.

    g. **Verify the health of the network.**
    Click the **Network Interfaces** tab and check the state of the host's network.

    If the network interface enters an "Out of sync" state or does not have an IP Address, click **Management → Refresh Capabilities**.

## 3.7. VERIFYING HEALING IN PROGRESS

After replacing a failed host with a new host, verify that your storage is healing as expected.

### Procedure

- **Verify that healing is in progress.**
  Run the following command on any hyperconverged host:

```
# for vol in `gluster volume list`; do gluster volume heal $vol info summary; done
```

  The output shows a summary of healing activity on each brick in each volume, for example:

```
Brick brick1
Status: Connected
Total Number of entries: 3
Number of entries in heal pending: 2
Number of entries in split-brain: 1
Number of entries possibly healing: 0
```

  Depending on brick size, volumes can take a long time to heal. You can still run and migrate virtual machines using this node while the underlying storage heals.

# CHAPTER 4. REUSING BRICKS AND RECONSTRUCTING EXISTING BRICK CONFIGURATION

## 4.1. HOST REPLACEMENT PREREQUISITES

- Determine which node to use as the Ansible controller node (the node from which all Ansible playbooks are executed). Red Hat recommends using a healthy node in the same cluster as the failed node as the Ansible controller node.
- If the failed host used Network-Bound Disk Encryption, ensure that you know the passphrase used for the existing disks.
- Take note of the disks that comprise the gluster volumes hosted by the server you are replacing.
- If possible, locate a recent backup or create a new backup of the important files (disk configuration or inventory files). See Backing up important files for details.
- Stop brick processes and unmount file systems on the failed host, to avoid file system inconsistency issues.

  ```
  # pkill glusterfsd
  # umount /gluster_bricks/{engine,vmstore,data}
  ```

- Check which operating system is running on your hyperconverged hosts by running the following command:

  ```
  $ nodectl info
  ```

- Reinstall the same operating system on the failed hyperconverged host.

## 4.2. PREPARING THE CLUSTER FOR HOST REPLACEMENT

1. **Verify host state in the Administrator Portal.**
   a. Log in to the Red Hat Virtualization Administrator Portal.
      The host is listed as **NonResponsive** in the Administrator Portal. Virtual machines that previously ran on this host are in the **Unknown** state.
   b. Click **Compute → Hosts** and click the Action menu ( ⋮ ).
   c. Click **Confirm host has been rebooted** and confirm the operation.
   d. Verify that the virtual machines are now listed with a state of **Down**.
2. **Update the SSH fingerprint for the failed node.**
   a. Log in to the Ansible controller node as the root user.
   b. Remove the existing SSH fingerprint for the failed node.

      ```
      # sed -i `/failed-host-frontend.example.com/d` /root/.ssh/known_hosts
      # sed -i `/failed-host-backend.example.com/d` /root/.ssh/known_hosts
      ```

   c. Copy the public key from the Ansible controller node to the freshly installed node.

```
# ssh-copy-id root@new-host-backend.example.com
# ssh-copy-id root@new-host-frontend.example.com
```

d. Verify that you can log in to all hosts in the cluster, including the Ansible controller node, using key-based SSH authentication without a password. Test access using all network addresses. The following example assumes that the Ansible controller node is **host1**.

```
# ssh root@host1-backend.example.com
# ssh root@host1-frontend.example.com
# ssh root@host2-backend.example.com
# ssh root@host2-frontend.example.com
# ssh root@new-host-backend.example.com
# ssh root@new-host-frontend.example.com
```

Use **ssh-copy-id** to copy the public key to any host you cannot log into without a password using this method.

```
# ssh-copy-id root@host-frontend.example.com
# ssh-copy-id root@host-backend.example.com
```

## 4.3. RECREATING DISK CONFIGURATION WITHOUT BACKUPS

If you do not have backup configuration files available for your cluster, you can recreate configuration using the following sections to ensure you are still able to use existing bricks and their data.

### 4.3.1. Reconfiguring encryption during host replacement

If the failed host used encryption, but you do not have backup encryption configuration available, you need to recreate your encryption configuration when you replace a failed host. Follow these steps to create encryption configuration files on the replacement host to match the other hosts in your existing cluster.

#### Procedure

1. **Set new keys and key files.**

   a. Store the passphrase for the LUKS encrypted disk in a temporary file in the **/root** directory.

   ```
   # echo passphrase /root/key
   ```

   If each disk has a separate passphrase, save them separately.

   ```
   # echo passphraseA /root/sda_key
   # echo passphraseB /root/sdb_key
   # echo passphraseC /root/sdc_key
   # echo passphraseD /root/sdd_key
   ```

   b. Generate new key files.

      i. Generate a random key file for each disk.

      ```
      # for disk in sda sdb sdc sdd; do dd if=/dev/urandom of=/etc/${disk}_keyfile bs=1024
      count=8192
      ```

ii. Set appropriate permissions on the new keyfiles.

```
# chown 400 /etc/*_keyfile
```

c. Set the new key for each disk.

```
# cryptsetup luksAddKey /etc/sda_keyfile --key-file /root/sda_key
# cryptsetup luksAddKey /etc/sdb_keyfile --key-file /root/sdb_key
# cryptsetup luksAddKey /etc/sdc_keyfile --key-file /root/sdc_key
# cryptsetup luksAddKey /etc/sdd_keyfile --key-file /root/sdd_key
```

2. **Verify each device can be opened with its key file.**

a. Determine the LUKS UUID for each device.

```
# cryptsetup luksUUID /dev/sdX
```

b. Open each device using its key file and UUID.

```
# cryptsetup luksOpen UUID=sdX-UUID luks_sdX -d /etc/sdX_keyfile
```

For example:

```
# cryptsetup luksOpen UUID=a28a19c7-6028-44df-b0b8-e5245944710c luks_sda -d
/etc/sda_keyfile
```

3. **Configure automatic decryption at boot time.**
Add a line for each device to the **/etc/crypttab** file using the following format.

```
# echo luks_sdX UUID=sdX-UUID /etc/sdX_keyfile >> /etc/crypttab
```

For example:

```
# echo luks_sda UUID=a28a19c7-6028-44df-b0b8-e5245944710c /etc/sda_keyfile >>
/etc/crypttab
```

4. **Set up Network-Bound Disk Encryption on the root disk.**

a. Change into the **hc-ansible-deployment** directory:

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
```

b. Create the inventory file.

i. Make a copy of the **luks_tang_inventory.yml** file for future reference.

```
cp luks_tang_inventory.yml luks_tang_inventory.yml.backup
```

ii. Define your configuration in the luks_tang_inventory.yml file.
Use the example **luks_tang_inventory.yml** file to define the details of disk encryption on each host. A complete outline of this file is available in Understanding the luks_tang_inventory.yml file.

c. Encrypt the **luks_tang_inventory.yml** file and specify a password using **ansible-vault**. The required variables in **luks_tang_inventory.yml** include password values, so it is important to encrypt the file to protect the password values.

```
# ansible-vault encrypt luks_tang_inventory.yml
```

Enter and confirm a new vault password when prompted.

d. Execute the **luks_tang_setup.yml** playbook with the **bindtang** tag.

```
# ansible-playbook -i luks_tang_inventory.yml tasks/luks_tang_setup.yml --
tags=bindtang --ask-vault-pass
```

Enter the vault password for this file when prompted to start disk encryption configuration.

### 4.3.2. Reconfiguring deduplication and compression during host replacement

If the failed host used deduplication and compression (VDO), but you do not have backup configuration information available, you need to recreate the deduplication and compression configuration when you replace a failed host. Follow these steps to create deduplication and compression configuration files on the replacement host to match the other hosts in your existing cluster.

**Procedure**

1. Copy the **/etc/vdoconf.yml** file from a healthy node to the replacement node.

```
# scp /etc/vdoconf.yml root@new-node.example.com:/etc/
```

2. Edit the indicated values in the **/etc/vdoconf.yml** file to provide the correct values for your replacement node.

IMPORTANT

Be careful when editing this file. Editing this file by hand is supported only when reconstructing deduplication and compression configuration without a backup file.

**vdo_sd***

Change this parameter to match the name of your VDO device.

**device**

Specify the VDO device using its **by-id** path. For normal volumes, this is something like **/dev/disk/by-id/scsi-xxx**. For encrypted volumes, this is something like **/dev/disk/by-id/dm-uuid-CRYPT-LUKS2-xxxxx**.

For example:

```
# cat /etc/vdoconf.yml

config: !Configuration
 vdos:
   vdo_sdc: !VDOService
     ...
```

```
    device: /dev/disk/by-id/scsi-360030480197f830125618adb17bac04c
    ...
    logicalSize: 180T
    ...
    physicalSize: 18625G
    ...
```

3. Restart the VDO service.

```
# systemctl restart vdo.service
```

### 4.3.3. Restoring disk mount configuration

If you do not have backup disk mount configuration, you need to recreate your configuration when you replace a host. Follow these steps to reconstruct disk mount configuration.

**Procedure**

1. Scan existing physical volumes, volume groups, and logical volumes.

```
# pvscan
# vgscan
# lvscan
```

2. Determine the UUID of each gluster brick.

```
# blkid lv_name
```

3. Add a line to the **/etc/fstab** file for each gluster brick, using the UUID.

```
# echo "UUID=64dfd1b1-4333-4ef6-8835-1053c6904d93 /gluster_bricks/engine xfs
inode64,noatime,nodiratime,_netdev,x-systemd.device-timeout=0 0 0" >> /etc/fstab
```

Volumes that use deduplication and compression need additional mount options, as shown:

```
# echo "UUID=64dfd1b1-4333-4ef6-8835-1053c6904d93 /gluster_bricks/vmstore xfs
inode64,noatime,nodiratime,_netdev,x-systemd.device-timeout=0,x-
systemd.requires=vdo.service 0 0" >> /etc/fstab
```

4. Create mount directories based on information from volumes.

```
# mkdir -p /gluster_bricks/{engine,vmstore,data}
```

5. Mount all bricks.

```
# mount -a
```

6. Set the required SELinux labels on all brick mount points.

```
# semanage fcontext -a -t glusterd_brick_t /gluster_bricks/engine
# semanage fcontext -a -t glusterd_brick_t /gluster_bricks/vmstore
# semanage fcontext -a -t glusterd_brick_t /gluster_bricks/data
# restorecon -Rv /gluster_bricks/engine
# restorecon -Rv /gluster_bricks/vmstore
# restorecon -Rv /gluster_bricks/data
```

## 4.4. CREATING THE NODE_PREP_INVENTORY.YML FILE

Define the replacement node in the **node_prep_inventory.yml** file.

### Procedure

1. Familiarize yourself with your Gluster configuration.
   The configuration that you define in your inventory file must match the existing Gluster volume configuration. Use **gluster volume info** to check where your bricks should be mounted for each Gluster volume, for example:

   ```
   # gluster volume info engine | grep -i brick
   Number of Bricks: 1 x 3 = 3
   Bricks:
   Brick1: host1.example.com:/gluster_bricks/engine/engine
   Brick2: host2.example.com:/gluster_bricks/engine/engine
   Brick3: host3.example.com:/gluster_bricks/engine/engine
   ```

2. Back up the **node_prep_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp node_prep_inventory.yml node_prep_inventory.yml.bk
   ```

3. Edit the **node_prep_inventory.yml** file to define your node preparation.
   See Appendix B, *Understanding the **node_prep_inventory.yml** file* for more information about this inventory file and its parameters.

## 4.5. CREATING THE NODE_REPLACE_INVENTORY.YML FILE

Define your cluster hosts by creating a **node_replacement_inventory.yml** file.

### Procedure

1. Back up the **node_replace_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp node_replace_inventory.yml node_replace_inventory.yml.bk
   ```

2. Edit the **node_replace_inventory.yml** file to define your cluster.
   See Appendix C, *Understanding the **node_replace_inventory.yml** file* for more information about this inventory file and its parameters.

## 4.6. EXECUTING THE REPLACE_NODE.YML PLAYBOOK FILE

The **replace_node.yml** playbook reconfigures a Red Hat Hyperconverged Infrastructure for Virtualization cluster to use a new node after an existing cluster node has failed.

**Procedure**

1. Execute the playbook.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/
# ansible-playbook -i node_prep_inventory.yml -i node_replace_inventory.yml
tasks/replace_node.yml
```

## 4.7. FINALIZING HOST REPLACEMENT

After you have replaced a failed host with a new host, follow these steps to ensure that the cluster is connected to the new host and properly activated.

**Procedure**

1. **Activate the host.**
    a. Log in to the Red Hat Virtualization Administrator Portal.
    b. Click **Compute → Hosts** and observe that the replacement host is listed with a state of **Maintenance**.
    c. Select the host and click **Management → Activate**.
    d. Wait for the host to reach the **Up** state.
2. **Attach the gluster network to the host.**
    a. Click **Compute → Hosts** and select the host.
    b. Click **Network Interfaces → Setup Host Networks**.
    c. Drag and drop the newly created network to the correct interface.
    d. Ensure that the **Verify connectivity between Host and Engine** checkbox is checked.
    e. Ensure that the **Save network configuration** checkbox is checked.
    f. Click **OK** to save.
    g. **Verify the health of the network.**
    Click the **Network Interfaces** tab and check the state of the host’s network.

        If the network interface enters an "Out of sync" state or does not have an IP Address, click **Management → Refresh Capabilities**.

## 4.8. VERIFYING HEALING IN PROGRESS

After replacing a failed host with a new host, verify that your storage is healing as expected.

**Procedure**

- **Verify that healing is in progress.**
  Run the following command on any hyperconverged host:

```
# for vol in `gluster volume list`; do gluster volume heal $vol info summary; done
```

The output shows a summary of healing activity on each brick in each volume, for example:

```
Brick brick1
Status: Connected
Total Number of entries: 3
Number of entries in heal pending: 2
Number of entries in split-brain: 1
Number of entries possibly healing: 0
```

Depending on brick size, volumes can take a long time to heal. You can still run and migrate virtual machines using this node while the underlying storage heals.

# CHAPTER 5. CREATING NEW BRICKS AND CONFIGURATION

## 5.1. HOST REPLACEMENT PREREQUISITES

- Determine which node to use as the Ansible controller node (the node from which all Ansible playbooks are executed). Red Hat recommends using a healthy node in the same cluster as the failed node as the Ansible controller node.
- Stop brick processes and unmount file systems on the failed host, to avoid file system inconsistency issues.

  ```
  # pkill glusterfsd
  # umount /gluster_bricks/{engine,vmstore,data}
  ```

- Check which operating system is running on your hyperconverged hosts by running the following command:

  ```
  $ nodectl info
  ```

- Reinstall the same operating system on the failed hyperconverged host.

## 5.2. PREPARING THE CLUSTER FOR HOST REPLACEMENT

1. **Verify host state in the Administrator Portal.**
   a. Log in to the Red Hat Virtualization Administrator Portal.
      The host is listed as **NonResponsive** in the Administrator Portal. Virtual machines that previously ran on this host are in the **Unknown** state.
   b. Click **Compute → Hosts** and click the Action menu ( ⋮ ).
   c. Click **Confirm host has been rebooted** and confirm the operation.
   d. Verify that the virtual machines are now listed with a state of **Down**.
2. **Update the SSH fingerprint for the failed node.**
   a. Log in to the Ansible controller node as the root user.
   b. Remove the existing SSH fingerprint for the failed node.

      ```
      # sed -i `/failed-host-frontend.example.com/d` /root/.ssh/known_hosts
      # sed -i `/failed-host-backend.example.com/d` /root/.ssh/known_hosts
      ```

   c. Copy the public key from the Ansible controller node to the freshly installed node.

      ```
      # ssh-copy-id root@new-host-backend.example.com
      # ssh-copy-id root@new-host-frontend.example.com
      ```

   d. Verify that you can log in to all hosts in the cluster, including the Ansible controller node, using key-based SSH authentication without a password. Test access using all network addresses. The following example assumes that the Ansible controller node is **host1**.

```
# ssh root@host1-backend.example.com
# ssh root@host1-frontend.example.com
# ssh root@host2-backend.example.com
# ssh root@host2-frontend.example.com
# ssh root@new-host-backend.example.com
# ssh root@new-host-frontend.example.com
```

Use **ssh-copy-id** to copy the public key to any host you cannot log into without a password using this method.

```
# ssh-copy-id root@host-frontend.example.com
# ssh-copy-id root@host-backend.example.com
```

## 5.3. CREATING THE NODE_PREP_INVENTORY.YML FILE

Define the replacement node in the **node_prep_inventory.yml** file.

### Procedure

1. Familiarize yourself with your Gluster configuration.
   The configuration that you define in your inventory file must match the existing Gluster volume configuration. Use **gluster volume info** to check where your bricks should be mounted for each Gluster volume, for example:

   ```
   # gluster volume info engine | grep -i brick
   Number of Bricks: 1 x 3 = 3
   Bricks:
   Brick1: host1.example.com:/gluster_bricks/engine/engine
   Brick2: host2.example.com:/gluster_bricks/engine/engine
   Brick3: host3.example.com:/gluster_bricks/engine/engine
   ```

2. Back up the **node_prep_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp node_prep_inventory.yml node_prep_inventory.yml.bk
   ```

3. Edit the **node_prep_inventory.yml** file to define your node preparation.
   See Appendix B, *Understanding the **node_prep_inventory.yml** file* for more information about this inventory file and its parameters.

## 5.4. CREATING THE NODE_REPLACE_INVENTORY.YML FILE

Define your cluster hosts by creating a **node_replacement_inventory.yml** file.

### Procedure

1. Back up the **node_replace_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp node_replace_inventory.yml node_replace_inventory.yml.bk
   ```

2. Edit the **node_replace_inventory.yml** file to define your cluster.

See Appendix C, *Understanding the* ***node_replace_inventory.yml*** *file* for more information about this inventory file and its parameters.

## 5.5. EXECUTING THE REPLACE_NODE.YML PLAYBOOK FILE

The **replace_node.yml** playbook reconfigures a Red Hat Hyperconverged Infrastructure for Virtualization cluster to use a new node after an existing cluster node has failed.

### Procedure

1. Execute the playbook.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/
# ansible-playbook -i node_prep_inventory.yml -i node_replace_inventory.yml
tasks/replace_node.yml
```

## 5.6. FINALIZING HOST REPLACEMENT

After you have replaced a failed host with a new host, follow these steps to ensure that the cluster is connected to the new host and properly activated.

### Procedure

1. **Activate the host.**
   a. Log in to the Red Hat Virtualization Administrator Portal.
   b. Click **Compute → Hosts** and observe that the replacement host is listed with a state of **Maintenance**.
   c. Select the host and click **Management → Activate**.
   d. Wait for the host to reach the **Up** state.
2. **Attach the gluster network to the host.**
   a. Click **Compute → Hosts** and select the host.
   b. Click **Network Interfaces → Setup Host Networks**.
   c. Drag and drop the newly created network to the correct interface.
   d. Ensure that the **Verify connectivity between Host and Engine** checkbox is checked.
   e. Ensure that the **Save network configuration** checkbox is checked.
   f. Click **OK** to save.
   g. **Verify the health of the network.**
      Click the **Network Interfaces** tab and check the state of the host’s network.

      If the network interface enters an "Out of sync" state or does not have an IP Address, click **Management → Refresh Capabilities**.

## 5.7. VERIFYING HEALING IN PROGRESS

After replacing a failed host with a new host, verify that your storage is healing as expected.

### Procedure

- **Verify that healing is in progress.**
  Run the following command on any hyperconverged host:

  ```
  # for vol in `gluster volume list`; do gluster volume heal $vol info summary; done
  ```

  The output shows a summary of healing activity on each brick in each volume, for example:

  ```
  Brick brick1
  Status: Connected
  Total Number of entries: 3
  Number of entries in heal pending: 2
  Number of entries in split-brain: 1
  Number of entries possibly healing: 0
  ```

  Depending on brick size, volumes can take a long time to heal. You can still run and migrate virtual machines using this node while the underlying storage heals.

# PART II. USING A DIFFERENT HOST FQDN

# CHAPTER 6. REPLACING A PRIMARY HOST USING NEW BRICKS

## 6.1. HOST REPLACEMENT PREREQUISITES

- Determine which node to use as the Ansible controller node (the node from which all Ansible playbooks are executed). Red Hat recommends using a healthy node in the same cluster as the failed node as the Ansible controller node.
- Power off all virtual machines in the cluster.
- Stop brick processes and unmount file systems on the failed host, to avoid file system inconsistency issues.

```
# pkill glusterfsd
# umount /gluster_bricks/{engine,vmstore,data}
```

- Check which operating system is running on your hyperconverged hosts by running the following command:

```
$ nodectl info
```

- Install the same operating system on a replacement host.

## 6.2. PREPARING THE CLUSTER FOR HOST REPLACEMENT

1. **Verify host state in the Administrator Portal.**
   a. Log in to the Red Hat Virtualization Administrator Portal.
      The host is listed as **NonResponsive** in the Administrator Portal. Virtual machines that previously ran on this host are in the **Unknown** state.
   b. Click **Compute → Hosts** and click the Action menu ( ⋮ ).
   c. Click **Confirm host has been rebooted** and confirm the operation.
   d. Verify that the virtual machines are now listed with a state of **Down**.
2. **Update the SSH fingerprint for the failed node.**
   a. Log in to the Ansible controller node as the root user.
   b. Remove the existing SSH fingerprint for the failed node.

```
# sed -i `/failed-host-frontend.example.com/d` /root/.ssh/known_hosts
# sed -i `/failed-host-backend.example.com/d` /root/.ssh/known_hosts
```

   c. Copy the public key from the Ansible controller node to the freshly installed node.

```
# ssh-copy-id root@new-host-backend.example.com
# ssh-copy-id root@new-host-frontend.example.com
```

d. Verify that you can log in to all hosts in the cluster, including the Ansible controller node, using key-based SSH authentication without a password. Test access using all network addresses. The following example assumes that the Ansible controller node is **host1**.

```
# ssh root@host1-backend.example.com
# ssh root@host1-frontend.example.com
# ssh root@host2-backend.example.com
# ssh root@host2-frontend.example.com
# ssh root@new-host-backend.example.com
# ssh root@new-host-frontend.example.com
```

Use **ssh-copy-id** to copy the public key to any host you cannot log into without a password using this method.

```
# ssh-copy-id root@host-frontend.example.com
# ssh-copy-id root@host-backend.example.com
```

## 6.3. CREATING THE NODE_PREP_INVENTORY.YML FILE

Define the replacement node in the **node_prep_inventory.yml** file.

### Procedure

1. Familiarize yourself with your Gluster configuration.
   The configuration that you define in your inventory file must match the existing Gluster volume configuration. Use **gluster volume info** to check where your bricks should be mounted for each Gluster volume, for example:

```
# gluster volume info engine | grep -i brick
Number of Bricks: 1 x 3 = 3
Bricks:
Brick1: host1.example.com:/gluster_bricks/engine/engine
Brick2: host2.example.com:/gluster_bricks/engine/engine
Brick3: host3.example.com:/gluster_bricks/engine/engine
```

2. Back up the **node_prep_inventory.yml** file.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
# cp node_prep_inventory.yml node_prep_inventory.yml.bk
```

3. Edit the **node_prep_inventory.yml** file to define your node preparation.
   See Appendix B, *Understanding the* ***node_prep_inventory.yml*** *file* for more information about this inventory file and its parameters.

## 6.4. CREATING THE NODE_REPLACE_INVENTORY.YML FILE

Define your cluster hosts by creating a **node_replacement_inventory.yml** file.

### Procedure

1. Back up the **node_replace_inventory.yml** file.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
# cp node_replace_inventory.yml node_replace_inventory.yml.bk
```

2. Edit the **node_replace_inventory.yml** file to define your cluster.
   See Appendix C, *Understanding the **node_replace_inventory.yml** file* for more information about this inventory file and its parameters.

## 6.5. EXECUTING THE REPLACE_NODE.YML PLAYBOOK FILE

The **replace_node.yml** playbook reconfigures a Red Hat Hyperconverged Infrastructure for Virtualization cluster to use a new node after an existing cluster node has failed.

### Procedure

1. Execute the playbook.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/
# ansible-playbook -i node_prep_inventory.yml -i node_replace_inventory.yml
tasks/replace_node.yml
```

## 6.6. UPDATING THE CLUSTER FOR A NEW PRIMARY HOST

When you replace a failed host using a different FQDN, you need to update configuration in the cluster to use the replacement host.

### Procedure

1. Change into the **hc-ansible-deployment** directory.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/
```

2. Make a copy of the **reconfigure_storage_inventory.yml** file.

```
# cp reconfigure_storage_inventory.yml reconfigure_storage_inventory.yml.bk
```

3. Edit the **reconfigure_storage_inventory.yml** file to identify the following:

   **hosts**
   Two active hosts in the cluster that have been configured to host the Hosted Engine virtual machine.

   **gluster_maintenance_old_node**
   The backend network FQDN of the failed node.

   **gluster_maintenance_new_node**
   The backend network FQDN of the replacement node.

   **ovirt_engine_hostname**
   The FQDN of the Hosted Engine virtual machine.

   For example:

```
all:
```

```
hosts:
  host2-backend.example.com:
  host3-backend.example.com:

vars:
  gluster_maintenance_old_node: host1-backend.example.com
  gluster_maintenance_new_node: host4-backend.example.com
  ovirt_engine_hostname: engine.example.com
```

4. Execute the **reconfigure_he_storage.yml** playbook with your updated inventory file.

```
# ansible-playbook -i reconfigure_he_storage_inventory.yml
tasks/reconfigure_he_storage.yml
```

## 6.7. REMOVING A FAILED HOST FROM THE CLUSTER

When a replacement host is ready, remove the existing failed host from the cluster.

### Procedure

1. **Remove the failed host.**
   a. Log in into the Administrator Portal.
   b. Click **Compute → Hosts**.
      The failed host is in the **NonResponsive** state. Virtual machines running on the failed host are in the **Unknown** state.
   c. Select the failed host.
   d. Click the main **Action menu ( ⋮ )** for the Hosts page and select **Confirm host has been rebooted**.
   e. Click **OK** to confirm the operation.
      Virtual machines move to the **Down** state.
   f. Select the failed host and click **Management → Maintenance**.
   g. Click the **Action menu ( ⋮ )** beside the failed host and click **Remove**.
2. **Update the storage domains.**
   For each storage domain:
   a. Click **Storage → Domains**.
   b. Click the storage domain name, then click **Data Center → Maintenance** and confirm the operation.
   c. Click **Manage Domain**.
      i. Edit the **Path** field to match the new FQDN.
      ii. Click **OK**.

NOTE

A dialog box with an **Operation Cancelled** error appears as a result of Bug 1853995, but the path is updated as expected.

d. Click the Action menu ( ⋮ ) beside the storage domain and click **Activate**.

3. **Add the replacement host to the cluster.**
4. **Attach the gluster logical network to the replacement host.**
5. **Restart all virtual machines.**
    a. For highly available virtual machines, disable and re-enable high-availability.
        i. Click **Compute → Virtual Machines** and select a virtual machine.
        ii. Click **Edit → High Availability** → uncheck the **High Availability** check box and click **OK**.
        iii. Click **Edit → High Availability** → check the **High Availability** check box and click **OK**.
    b. Start all the virtual machines.
        i. Click **Compute → Virtual Machines** and select a virtual machine.
        ii. Click the **Action menu ( ⋮ ) → Start**.

## 6.8. VERIFYING HEALING IN PROGRESS

After replacing a failed host with a new host, verify that your storage is healing as expected.

**Procedure**

- **Verify that healing is in progress.**
  Run the following command on any hyperconverged host:

```
# for vol in `gluster volume list`; do gluster volume heal $vol info summary; done
```

  The output shows a summary of healing activity on each brick in each volume, for example:

```
Brick brick1
Status: Connected
Total Number of entries: 3
Number of entries in heal pending: 2
Number of entries in split-brain: 1
Number of entries possibly healing: 0
```

  Depending on brick size, volumes can take a long time to heal. You can still run and migrate virtual machines using this node while the underlying storage heals.

# CHAPTER 7. REPLACING A NON-PRIMARY HOST USING NEW BRICKS

## 7.1. HOST REPLACEMENT PREREQUISITES

- Determine which node to use as the Ansible controller node (the node from which all Ansible playbooks are executed). Red Hat recommends using a healthy node in the same cluster as the failed node as the Ansible controller node.
- Stop brick processes and unmount file systems on the failed host, to avoid file system inconsistency issues.

```
# pkill glusterfsd
# umount /gluster_bricks/{engine,vmstore,data}
```

- Check which operating system is running on your hyperconverged hosts by running the following command:

```
$ nodectl info
```

- Install the same operating system on a replacement host.

## 7.2. PREPARING THE CLUSTER FOR HOST REPLACEMENT

1. **Verify host state in the Administrator Portal.**
   a. Log in to the Red Hat Virtualization Administrator Portal.
      The host is listed as **NonResponsive** in the Administrator Portal. Virtual machines that previously ran on this host are in the **Unknown** state.
   b. Click **Compute → Hosts** and click the Action menu ( ⋮ ).
   c. Click **Confirm host has been rebooted** and confirm the operation.
   d. Verify that the virtual machines are now listed with a state of **Down**.
2. **Update the SSH fingerprint for the failed node.**
   a. Log in to the Ansible controller node as the root user.
   b. Remove the existing SSH fingerprint for the failed node.

```
# sed -i `/failed-host-frontend.example.com/d` /root/.ssh/known_hosts
# sed -i `/failed-host-backend.example.com/d` /root/.ssh/known_hosts
```

   c. Copy the public key from the Ansible controller node to the freshly installed node.

```
# ssh-copy-id root@new-host-backend.example.com
# ssh-copy-id root@new-host-frontend.example.com
```

   d. Verify that you can log in to all hosts in the cluster, including the Ansible controller node, using key-based SSH authentication without a password. Test access using all network addresses. The following example assumes that the Ansible controller node is **host1**.

```
# ssh root@host1-backend.example.com
# ssh root@host1-frontend.example.com
# ssh root@host2-backend.example.com
# ssh root@host2-frontend.example.com
# ssh root@new-host-backend.example.com
# ssh root@new-host-frontend.example.com
```

Use **ssh-copy-id** to copy the public key to any host you cannot log into without a password using this method.

```
# ssh-copy-id root@host-frontend.example.com
# ssh-copy-id root@host-backend.example.com
```

## 7.3. CREATING THE NODE_PREP_INVENTORY.YML FILE

Define the replacement node in the **node_prep_inventory.yml** file.

### Procedure

1. Familiarize yourself with your Gluster configuration.
   The configuration that you define in your inventory file must match the existing Gluster volume configuration. Use **gluster volume info** to check where your bricks should be mounted for each Gluster volume, for example:

   ```
   # gluster volume info engine | grep -i brick
   Number of Bricks: 1 x 3 = 3
   Bricks:
   Brick1: host1.example.com:/gluster_bricks/engine/engine
   Brick2: host2.example.com:/gluster_bricks/engine/engine
   Brick3: host3.example.com:/gluster_bricks/engine/engine
   ```

2. Back up the **node_prep_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp node_prep_inventory.yml node_prep_inventory.yml.bk
   ```

3. Edit the **node_prep_inventory.yml** file to define your node preparation.
   See Appendix B, *Understanding the* ***node_prep_inventory.yml*** *file* for more information about this inventory file and its parameters.

## 7.4. CREATING THE NODE_REPLACE_INVENTORY.YML FILE

Define your cluster hosts by creating a **node_replacement_inventory.yml** file.

### Procedure

1. Back up the **node_replace_inventory.yml** file.

   ```
   # cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment
   # cp node_replace_inventory.yml node_replace_inventory.yml.bk
   ```

2. Edit the **node_replace_inventory.yml** file to define your cluster.

See Appendix C, *Understanding the* ***node_replace_inventory.yml*** *file* for more information about this inventory file and its parameters.

## 7.5. EXECUTING THE REPLACE_NODE.YML PLAYBOOK FILE

The **replace_node.yml** playbook reconfigures a Red Hat Hyperconverged Infrastructure for Virtualization cluster to use a new node after an existing cluster node has failed.

### Procedure

1. Execute the playbook.

```
# cd /etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/
# ansible-playbook -i node_prep_inventory.yml -i node_replace_inventory.yml
tasks/replace_node.yml
```

## 7.6. REMOVING A FAILED HOST FROM THE CLUSTER

When a replacement host is ready, remove the existing failed host from the cluster.

### Procedure

1. **Remove the failed host.**

   a. Log in to the Administrator Portal.

   b. Click **Compute → Hosts**.
   The replacement host is in the **NonResponsive** state. Virtual machines running on that host are in the **Unknown** state.

   c. Select the replacement host.

   d. Click the main **Action menu ( ⋮ )** for the Hosts page and select **Confirm host has been rebooted**.

   e. Click **OK** to confirm.

   f. Click the **Action menu ( ⋮ )** beside the failed host and click **Remove**.

2. **Clean stale Hosted Engine metadata.**

   a. Determine the identifier of the failed node.

```
# hosted-engine --vm-status | grep failed-node.example.com
--== Host server1-frontend.example.com (id: 1) status ==--
Hostname                           : failed-node.example.com
```

   b. Remove the metadata associated with that host identifier.

```
# hosted-engine --clean-metadata --host-id=1 --force
```

## 7.7. VERIFYING HEALING IN PROGRESS

After replacing a failed host with a new host, verify that your storage is healing as expected.

**Procedure**

- **Verify that healing is in progress.**
  Run the following command on any hyperconverged host:

  ```
  # for vol in `gluster volume list`; do gluster volume heal $vol info summary; done
  ```

  The output shows a summary of healing activity on each brick in each volume, for example:

  ```
  Brick brick1
  Status: Connected
  Total Number of entries: 3
  Number of entries in heal pending: 2
  Number of entries in split-brain: 1
  Number of entries possibly healing: 0
  ```

  Depending on brick size, volumes can take a long time to heal. You can still run and migrate virtual machines using this node while the underlying storage heals.

# PART III. REFERENCE MATERIAL

# APPENDIX A. UNDERSTANDING THE LUKS_TANG_INVENTORY.YML FILE

## A.1. CONFIGURATION PARAMETERS FOR DISK ENCRYPTION

### hc_nodes (required)

A list of hyperconverged hosts that uses the back-end FQDN of the host, and the configuration details of those hosts. Configuration that is specific to a host is defined under that host's back-end FQDN. Configuration that is common to all hosts is defined in the vars: section.

```
hc_nodes:
  hosts:
    host1backend.example.com:
      [configuration specific to this host]
    host2backend.example.com:
    host3backend.example.com:
    host4backend.example.com:
    host5backend.example.com:
    host6backend.example.com:
  vars:
    [configuration common to all hosts]
```

### blacklist_mpath_devices (optional)

By default, Red Hat Virtualization Host enables multipath configuration, which provides unique multipath names and worldwide identifiers for all disks, even when disks do not have underlying multipath configuration. Include this section if you do not have multipath configuration so that the multipath device names are not used for listed devices. Disks that are not listed here are assumed to have multipath configuration available, and require the path format **/dev/mapper/<WWID>** instead of **/dev/sdx** when defined in subsequent sections of the inventory file.
On a server with four devices (sda, sdb, sdc and sdd), the following configuration blacklists only two devices. The path format /dev/mapper/<WWID> is expected for devices not in this list.

```
hc_nodes:
  hosts:
    host1backend.example.com:
      blacklist_mpath_devices:
        - sdb
        - sdc
```

### gluster_infra_luks_devices (required)

A list of devices to encrypt and the encryption passphrase to use for each device.

```
hc_nodes:
  hosts:
    host1backend.example.com:
      gluster_infra_luks_devices:
        - devicename: /dev/sdb
          passphrase: Str0ngPa55#
```

### devicename

The name of the device in the format **/dev/sdx**.

### passphrase

The password to use for this device when configuring encryption. After disk encryption with Network-Bound Disk Encryption (NBDE) is configured, a new random key is generated, providing greater security.

**rootpassphrase (required)**

The password that you used when you selected **Encrypt my data** during operating system installation on this host.

```
hc_nodes:
  hosts:
    host1backend.example.com:
      rootpassphrase: h1-Str0ngPa55#
```

**rootdevice (required)**

The root device that was encrypted when you selected **Encrypt my data** during operating system installation on this host.

```
hc_nodes:
  hosts:
    host1backend.example.com:
      rootdevice: /dev/sda2
```

**networkinterface (required)**

The network interface this host uses to reach the NBDE key server.

```
hc_nodes:
  hosts:
    host1backend.example.com:
      networkinterface: ens3s0f0
```

**ip_version (required)**

Whether to use IPv4 or IPv6 networking. Valid values are **IPv4** and **IPv6**. There is no default value. Mixed networks are not supported.

```
hc_nodes:
  vars:
    ip_version: IPv4
```

**ip_config_method (required)**

Whether to use DHCP or static networking. Valid values are **dhcp** and **static**. There is no default value.

```
hc_nodes:
  vars:
    ip_config_method: dhcp
```

The other valid value for this option is **static**, which requires the following additional parameters and is defined individually for each host:

```
hc_nodes:
  hosts:
    host1backend.example.com:
      ip_config_method: static
      host_ip_addr: 192.168.1.101
      host_ip_prefix: 24
      host_net_gateway: 192.168.1.100
    host2backend.example.com:
      ip_config_method: static
      host_ip_addr: 192.168.1.102
      host_ip_prefix: 24
      host_net_gateway: 192.168.1.100
    host3backend.example.com:
      ip_config_method: static
      host_ip_addr: 192.168.1.102
      host_ip_prefix: 24
      host_net_gateway: 192.168.1.100
```

### gluster_infra_tangservers

The address of your NBDE key server or servers, including **http:**//. If your servers use a port other than the default (80), specify a port by appending **:_port_** to the end of the URL.

```
hc_nodes:
  vars:
    gluster_infra_tangservers:
      - url: http://key-server1.example.com
      - url: http://key-server2.example.com:80
```

## A.2. EXAMPLE LUKS_TANG_INVENTORY.YML

### Dynamically allocated IP addresses

```
hc_nodes:
  hosts:
    host1-backend.example.com:
      blacklist_mpath_devices:
        - sda
        - sdb
        - sdc
      gluster_infra_luks_devices:
        - devicename: /dev/sdb
          passphrase: dev-sdb-encrypt-passphrase
        - devicename: /dev/sdc
          passphrase: dev-sdc-encrypt-passphrase
      rootpassphrase: host1-root-passphrase
      rootdevice: /dev/sda2
      networkinterface: eth0
    host2-backend.example.com:
      blacklist_mpath_devices:
        - sda
        - sdb
        - sdc
      gluster_infra_luks_devices:
        - devicename: /dev/sdb
          passphrase: dev-sdb-encrypt-passphrase
        - devicename: /dev/sdc
        passphrase: dev-sdc-encrypt-passphrase
      rootpassphrase: host2-root-passphrase
      rootdevice: /dev/sda2
      networkinterface: eth0
    host3-backend.example.com:
      blacklist_mpath_devices:
        - sda
        - sdb
        - sdc
      gluster_infra_luks_devices:
        - devicename: /dev/sdb
          passphrase: dev-sdb-encrypt-passphrase
        - devicename: /dev/sdc
          passphrase: dev-sdc-encrypt-passphrase
      rootpassphrase: host3-root-passphrase
      rootdevice: /dev/sda2
      networkinterface: eth0
  vars:
    ip_version: IPv4
    ip_config_method: dhcp
    gluster_infra_tangservers:
      - url: http://key-server1.example.com:80
      - url: http://key-server2.example.com:80
```

### Static IP addresses

```
hc_nodes:
  hosts:
    host1-backend.example.com:
      blacklist_mpath_devices:
        - sda
        - sdb
        - sdc
      gluster_infra_luks_devices:
        - devicename: /dev/sdb
          passphrase: dev-sdb-encrypt-passphrase
        - devicename: /dev/sdc
          passphrase: dev-sdc-encrypt-passphrase
      rootpassphrase: host1-root-passphrase
      rootdevice: /dev/sda2
      networkinterface: eth0
      host_ip_addr: host1-static-ip
      host_ip_prefix: network-prefix
      host_net_gateway: default-network-gateway
    host2-backend.example.com:
      blacklist_mpath_devices:
        - sda
        - sdb
        - sdc
      gluster_infra_luks_devices:
        - devicename: /dev/sdb
          passphrase: dev-sdb-encrypt-passphrase
        - devicename: /dev/sdc
          passphrase: dev-sdc-encrypt-passphrase
      rootpassphrase: host2-root-passphrase
      rootdevice: /dev/sda2
      networkinterface: eth0
      host_ip_addr: host1-static-ip
      host_ip_prefix: network-prefix
      host_net_gateway: default-network-gateway
    host3-backend.example.com:
      blacklist_mpath_devices:
        - sda
        - sdb
        - sdc
      gluster_infra_luks_devices:
        - devicename: /dev/sdb
          passphrase: dev-sdb-encrypt-passphrase
        - devicename: /dev/sdc
          passphrase: dev-sdc-encrypt-passphrase
      rootpassphrase: host3-root-passphrase
      rootdevice: /dev/sda2
      networkinterface: eth0
      host_ip_addr: host1-static-ip
      host_ip_prefix: network-prefix
      host_net_gateway: default-network-gateway
  vars:
    ip_version: IPv4
    ip_config_method: static
    gluster_infra_tangservers:
      - url: http://key-server1.example.com:80
      - url: http://key-server2.example.com:80
```

# APPENDIX B. UNDERSTANDING THE NODE_PREP_INVENTORY.YML FILE

The **node_prep_inventory.yml** file is an example Ansible inventory file that you can use to prepare a replacement host for your Red Hat Hyperconverged Infrastructure for Virtualization cluster.

You can find this file at **/etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/node_prep_inventory.yml** on any hyperconverged host.

## B.1. CONFIGURATION PARAMETERS FOR PREPARING A REPLACEMENT NODE

### B.1.1. Hosts to configure

**hc_nodes**

A list of hyperconverged hosts that uses the back-end FQDN of the host, and the configuration details of those hosts. Configuration that is specific to a host is defined under that host’s back-end FQDN. Configuration that is common to all hosts is defined in the **vars:** section.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      [configuration specific to this host]
  vars:
    [configuration common to all hosts]
```

### B.1.2. Multipath devices

**blacklist_mpath_devices** (optional)

By default, Red Hat Virtualization Host enables multipath configuration, which provides unique multipath names and worldwide identifiers for all disks, even when disks do not have underlying multipath configuration. Include this section if you do not have multipath configuration so that the multipath device names are not used for listed devices. Disks that are not listed here are assumed to have multipath configuration available, and require the path format **/dev/mapper/<WWID>** instead of **/dev/sdx** when defined in subsequent sections of the inventory file.
On a server with four devices (**sda**, **sdb**, **sdc** and **sdd**), the following configuration blacklists only two devices. The path format **/dev/mapper/<WWID>** is expected for devices not in this list.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      blacklist_mpath_devices:
        - sdb
        - sdc
```

IMPORTANT

Do not list encrypted devices (**luks_*** devices) in **blacklist_mpath_devices**, as they require multipath configuration to work.

### B.1.3. Deduplication and compression

**gluster_infra_vdo** (optional)

Include this section to define a list of devices to use deduplication and compression. These devices require the **/dev/mapper/<name>** path format when you define them as volume groups in **gluster_infra_volume_groups**. Each device listed must have the following information:

**name**

A short name for the VDO device, for example **vdo_sdc**.

**device**

The device to use, for example, **/dev/sdc**.

**logicalsize**

The logical size of the VDO volume. Set this to ten times the size of the physical disk, for example, if you have a 500 GB disk, set **logicalsize: '5000G'**.

**emulate512**

If you use devices with a 4 KB block size, set this to **on**.

**slabsize**

If the logical size of the volume is 1000 GB or larger, set this to **32G**. If the logical size is smaller than 1000 GB, set this to **2G**.

**blockmapcachesize**

Set this to **128M**.

**writepolicy**

Set this to **auto**.

For example:

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      gluster_infra_vdo:
      - { name: 'vdo_sdc', device: '/dev/sdc', logicalsize: '5000G',
        emulate512: 'off', slabsize: '32G', blockmapcachesize: '128M',
        writepolicy: 'auto' }
      - { name: 'vdo_sdd', device: '/dev/sdd', logicalsize: '500G',
        emulate512: 'off', slabsize: '2G', blockmapcachesize: '128M',
        writepolicy: 'auto' }
```

### B.1.4. Storage infrastructure

**gluster_infra_volume_groups** (required)

This section creates the volume groups that contain the logical volumes.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      gluster_infra_volume_groups:
        - vgname: gluster_vg_sdb
          pvname: /dev/sdb
        - vgname: gluster_vg_sdc
          pvname: /dev/mapper/vdo_sdc
```

**gluster_infra_mount_devices** (required)

This section creates the logical volumes that form Gluster bricks.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      gluster_infra_mount_devices:
        - path: /gluster_bricks/engine
          lvname: gluster_lv_engine
          vgname: gluster_vg_sdb
        - path: /gluster_bricks/data
          lvname: gluster_lv_data
          vgname: gluster_vg_sdc
        - path: /gluster_bricks/vmstore
          lvname: gluster_lv_vmstore
          vgname: gluster_vg_sdd
```

**gluster_infra_thinpools** (optional)

This section defines logical thin pools for use by thinly provisioned volumes. Thin pools are not suitable for the **engine** volume, but can be used for the **vmstore** and **data** volume bricks.

**vgname**

The name of the volume group that contains this thin pool.

**thinpoolname**

A name for the thin pool, for example, **gluster_thinpool_sdc**.

**thinpoolsize**

The sum of the sizes of all logical volumes to be created in this volume group.

**poolmetadatasize**

Set to **16G**; this is the recommended size for supported deployments.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      gluster_infra_thinpools:
        - {vgname: 'gluster_vg_sdc', thinpoolname: 'gluster_thinpool_sdc', thinpoolsize: '500G',
poolmetadatasize: '16G'}
        - {vgname: 'gluster_vg_sdd', thinpoolname: 'gluster_thinpool_sdd', thinpoolsize: '500G',
poolmetadatasize: '16G'}
```

**gluster_infra_cache_vars** (optional)

This section defines cache logical volumes to improve performance for slow devices. A fast cache device is attached to a thin pool, and requires **gluster_infra_thinpool** to be defined.

**vgname**

The name of a volume group with a slow device that requires a fast external cache.

**cachedisk**

The paths of the slow and fast devices, separated with a comma, for example, to use a cache device **sde** with the slow device **sdb**, specify **/dev/sdb,/dev/sde**.

**cachelvname**

A name for this cache logical volume.

**cachethinpoolname**

The thin pool to which the fast cache volume is attached.

**cachelvsize**

The size of the cache logical volume. Around 0.01% of this size is used for cache metadata.

**cachemode**

The cache mode. Valid values are **writethrough** and **writeback**.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      gluster_infra_cache_vars:
        - vgname: gluster_vg_sdb
          cachedisk: /dev/sdb,/dev/sde
          cachelvname: cachelv_thinpool_sdb
          cachethinpoolname: gluster_thinpool_sdb
          cachelvsize: '250G'
          cachemode: writethrough
```

**gluster_infra_thick_lvs** (required)

The thickly provisioned logical volumes that are used to create bricks. Bricks for the **engine** volume must be thickly provisioned.

**vgname**

The name of the volume group that contains the logical volume.

**lvname**

The name of the logical volume.

**size**

The size of the logical volume. The **engine** logical volume requires **100G**.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      gluster_infra_thick_lvs:
        - vgname: gluster_vg_sdb
          lvname: gluster_lv_engine
          size: 100G
```

**gluster_infra_lv_logicalvols** (required)

The thinly provisioned logical volumes that are used to create bricks.

**vgname**

The name of the volume group that contains the logical volume.

**thinpool**

The thin pool that contains the logical volume, if this volume is thinly provisioned.

**lvname**

The name of the logical volume.

**size**

The size of the logical volume. The **engine** logical volume requires **100G**.

```
hc_nodes:
  hosts:
    new-host-backend-fqdn.example.com:
      gluster_infra_lv_logicalvols:
        - vgname: gluster_vg_sdc
          thinpool: gluster_thinpool_sdc
          lvname: gluster_lv_data
          lvsize: 200G
        - vgname: gluster_vg_sdd
          thinpool: gluster_thinpool_sdd
          lvname: gluster_lv_vmstore
          lvsize: 200G
```

**gluster_infra_disktype** (required)

Specifies the underlying hardware configuration of the disks. Set this to the value that matches your hardware: **RAID6**, **RAID5**, or **JBOD**.

```
hc_nodes:
  vars:
    gluster_infra_disktype: RAID6
```

**gluster_infra_diskcount** (required)

Specifies the number of data disks in the RAID set. For a **JBOD** disk type, set this to **1**.

```
hc_nodes:
  vars:
    gluster_infra_diskcount: 10
```

**gluster_infra_stripe_unit_size** (required)

The stripe size of the RAID set in megabytes.

```
hc_nodes:
  vars:
    gluster_infra_stripe_unit_size: 256
```

**gluster_features_force_varlogsizecheck** (required)

Set this to **true** if you want to verify that your **/var/log** partition has sufficient free space during the deployment process. It is important to have sufficient space for logs, but it is not required to verify space requirements at deployment time if you plan to monitor space requirements carefully.

```
hc_nodes:
  vars:
    gluster_features_force_varlogsizecheck: false
```

**gluster_set_selinux_labels** (required)

Ensures that volumes can be accessed when SELinux is enabled. Set this to **true** if SELinux is enabled on this host.

```
hc_nodes:
  vars:
    gluster_set_selinux_labels: true
```

### B.1.5. Firewall and network infrastructure

**gluster_infra_fw_ports** (required)

A list of ports to open between all nodes, in the format **<port>/<protocol>**.

```
hc_nodes:
  vars:
    gluster_infra_fw_ports:
       - 2049/tcp
       - 54321/tcp
       - 5900-6923/tcp
       - 16514/tcp
       - 5666/tcp
       - 16514/tcp
```

**gluster_infra_fw_permanent** (required)

Ensures the ports listed in **gluster_infra_fw_ports** are open after nodes are rebooted. Set this to **true** for production use cases.

```
hc_nodes:
  vars:
  gluster_infra_fw_permanent: true
```

**gluster_infra_fw_state** (required)

Enables the firewall. Set this to **enabled** for production use cases.

```
hc_nodes:
  vars:
    gluster_infra_fw_state: enabled
```

**gluster_infra_fw_zone** (required)

Specifies the firewall zone to which these **gluster_infra_fw_*** parameters are applied.

```
hc_nodes:
  vars:
    gluster_infra_fw_zone: public
```

**gluster_infra_fw_services** (required)

A list of services to allow through the firewall. Ensure **glusterfs** is defined here.

```
hc_nodes:
  vars:
  gluster_infra_fw_services:
     - glusterfs
```

## B.2. EXAMPLE NODE_PREP_INVENTORY.YML

```
# Section for Host Preparation Phase
hc_nodes:
```

```
hosts:
   # Host - The node which need to be prepared for replacement
  new-host-backend-fqdn.example.com:

    # Blacklist multipath devices which are used for gluster bricks
    # If you omit blacklist_mpath_devices it means all device will be whitelisted.
    # If the disks are not blacklisted, and then its taken that multipath configuration
    # exists in the server and one should provide /dev/mapper/<WWID> instead of /dev/sdx
    blacklist_mpath_devices:
      - sdb
      - sdc

    # Enable this section gluster_infra_vdo, if dedupe & compression is
    # required on that storage volume.
    # The variables refers to:
    # name        - VDO volume name to be used
    # device      - Disk name on which VDO volume to created
    # logicalsize - Logical size of the VDO volume.This value is 10 times
    #               the size of the physical disk
    # emulate512  - VDO device is made as 4KB block sized storage volume(4KN)
    # slabsize    - VDO slab size. If VDO logical size >= 1000G then
    #               slabsize is 32G else slabsize is 2G
    #
    # Following VDO values are as per recommendation and treated as constants:
    # blockmapcachesize - 128M
    # writepolicy       - auto
    #
    # gluster_infra_vdo:
    #   - { name: vdo_sdc, device: /dev/sdc, logicalsize: 5000G, emulate512: off, slabsize: 32G,
    #       blockmapcachesize:  128M, writepolicy: auto }
    #   - { name: vdo_sdd, device: /dev/sdd, logicalsize: 3000G, emulate512: off, slabsize: 32G,
    #       blockmapcachesize:  128M, writepolicy: auto }

    # When dedupe and compression is enabled on the device,
    # use pvname for that device as /dev/mapper/<vdo_device_name> # # The variables refers to: #
vgname - VG to be created on the disk # pvname - Physical disk (/dev/sdc) or VDO volume
(/dev/mapper/vdo_sdc) gluster_infra_volume_groups: - vgname: gluster_vg_sdb pvname: /dev/sdb -
vgname: gluster_vg_sdc pvname: /dev/mapper/vdo_sdc - vgname: gluster_vg_sdd pvname:
/dev/mapper/vdo_sdd gluster_infra_mount_devices: - path: /gluster_bricks/engine lvname:
gluster_lv_engine vgname: gluster_vg_sdb - path: /gluster_bricks/data lvname: gluster_lv_data
vgname: gluster_vg_sdc - path: /gluster_bricks/vmstore lvname: gluster_lv_vmstore vgname:
gluster_vg_sdd # 'thinpoolsize is the sum of sizes of all LVs to be created on that VG
    # In the case of VDO enabled, thinpoolsize is 10 times the sum of sizes
    # of all LVs to be created on that VG. Recommended values for
    # poolmetadatasize is 16GB and that should be considered exclusive of
    # thinpoolsize
    gluster_infra_thinpools:
      - {vgname: gluster_vg_sdc, thinpoolname: gluster_thinpool_sdc, thinpoolsize: 500G,
poolmetadatasize: 16G}
      - {vgname: gluster_vg_sdd, thinpoolname: gluster_thinpool_sdd, thinpoolsize: 500G,
poolmetadatasize: 16G}

    # Enable the following section if LVM cache is to enabled
    # Following are the variables:
    # vgname           - VG with the slow HDD device that needs caching
    # cachedisk        - Comma separated value of slow HDD and fast SSD
```

```
      #                    In this example, /dev/sdb is the slow HDD, /dev/sde is fast SSD
      # cachelvname       - LV cache name
      # cachethinpoolname - Thinpool to which the fast SSD to be attached
      # cachelvsize       - Size of cache data LV. This is the SSD_size - (1/1000) of SSD_size
      #                    1/1000th of SSD space will be used by cache LV meta
      # cachemode         - writethrough or writeback
      # gluster_infra_cache_vars:
      #  - vgname: gluster_vg_sdb
      #    cachedisk: /dev/sdb,/dev/sde
      #    cachelvname: cachelv_thinpool_sdb
      #    cachethinpoolname: gluster_thinpool_sdb
      #    cachelvsize: 250G
      #    cachemode: writethrough

      # Only the engine brick needs to be thickly provisioned
      # Engine brick requires 100GB of disk space
      gluster_infra_thick_lvs:
        - vgname: gluster_vg_sdb
          lvname: gluster_lv_engine
          size: 100G

      gluster_infra_lv_logicalvols:
        - vgname: gluster_vg_sdc
          thinpool: gluster_thinpool_sdc
          lvname: gluster_lv_data
          lvsize: 200G
        - vgname: gluster_vg_sdd
          thinpool: gluster_thinpool_sdd
          lvname: gluster_lv_vmstore
          lvsize: 200G

  # Common configurations
  vars:
    # In case of IPv6 based deployment "gluster_features_enable_ipv6" needs to be enabled,below
line needs to be uncommented, like:
    # gluster_features_enable_ipv6: true

    # Firewall setup
    gluster_infra_fw_ports:
       - 2049/tcp
       - 54321/tcp
       - 5900-6923/tcp
       - 16514/tcp
       - 5666/tcp
       - 16514/tcp
    gluster_infra_fw_permanent: true
    gluster_infra_fw_state: enabled
    gluster_infra_fw_zone: public
    gluster_infra_fw_services:
       - glusterfs
    # Allowed values for gluster_infra_disktype - RAID6, RAID5, JBOD
    gluster_infra_disktype: RAID6

    # gluster_infra_diskcount is the number of data disks in the RAID set.
    #  Note for JBOD its 1
    gluster_infra_diskcount: 10
```

```
gluster_infra_stripe_unit_size: 256
gluster_features_force_varlogsizecheck: false
gluster_set_selinux_labels: true
```

# APPENDIX C. UNDERSTANDING THE NODE_REPLACE_INVENTORY.YML FILE

The **node_replace_inventory.yml** file is an example Ansible inventory file that you can use to prepare a replacement host for your Red Hat Hyperconverged Infrastructure for Virtualization cluster.

You can find this file at **/etc/ansible/roles/gluster.ansible/playbooks/hc-ansible-deployment/node_replace_inventory.yml** on any hyperconverged host.

## C.1. CONFIGURATION PARAMETERS FOR NODE REPLACEMENT

**hosts (required)**

Defines one active host in the cluster using the back-end FQDN.

```
cluster_nodes:
  hosts:
    host2-backend-fqdn.example.com:
  vars:
    [common host configuration]
```

**gluster_maintenance_old_node (required)**

Defines the backend FQDN of the node being replaced.

```
cluster_nodes:
  hosts:
    host2-backend-fqdn.example.com:
  vars:
    gluster_maintenance_old_node: host1-backend-fqdn.example.com
```

**gluster_maintenance_new_node (required)**

Defines the backend FQDN of the replacement node.

```
cluster_nodes:
  hosts:
    host2-backend-fqdn.example.com:
  vars:
    gluster_maintenance_new_node: new-host-backend-fqdn.example.com
```

**gluster_maintenance_cluster_node (required)**

An active node in the cluster. Cannot be the same as **gluster_maintenance_cluster_node_2**.

```
cluster_nodes:
  hosts:
    host2-backend-fqdn.example.com:
  vars:
    gluster_maintenance_cluster_node: host2-backend-fqdn.example.com
```

**gluster_maintenance_cluster_node_2 (required)**

An active node in the cluster. Cannot be the same as **gluster_maintenance_cluster_node**.

```
cluster_nodes:
  hosts:
    host2-backend-fqdn.example.com:
  vars:
    gluster_maintenance_cluster_node_2: host3-backend-fqdn.example.com
```

## C.2. EXAMPLE NODE_REPLACE_INVENTORY.YML

```
cluster_node:
  hosts:
   host2-backend-fqdn.example.com:

  vars:
    gluster_maintenance_old_node: host1-backend-fqdn.example.com
    gluster_maintenance_new_node: new-host-backend-fqdn.example.com
    gluster_maintenance_cluster_node: host2-backend-fqdn.example.com
    gluster_maintenance_cluster_node_2: host3-backend-fqdn.example.com
```