


High Availability

Table of Contents

Table of Contents	1
Overview	4
Software and Hardware Requirements	4
Known Issues and Limitations	5
Topics in this section	5
Configuring Nodes	6
What is a Node?	6
How many Nodes will you need?	6
What data is replicated between Nodes?	7
Configuring Blob Stores	7
What is a Blob Store?	7
Choosing a Type for your Blob Stores	8
Recommended Shared File Systems	8
Cloud Object Stores	9
File Systems to avoid	10
Determining a Backup Solution	11
Configuring Hazelcast	11
Network Preparation	11
Node Discovery	12
Multicast Discovery	12
AWS Discovery	13
TCP/IP Discovery	15
Designing your Cluster Backup/Restore Process	16

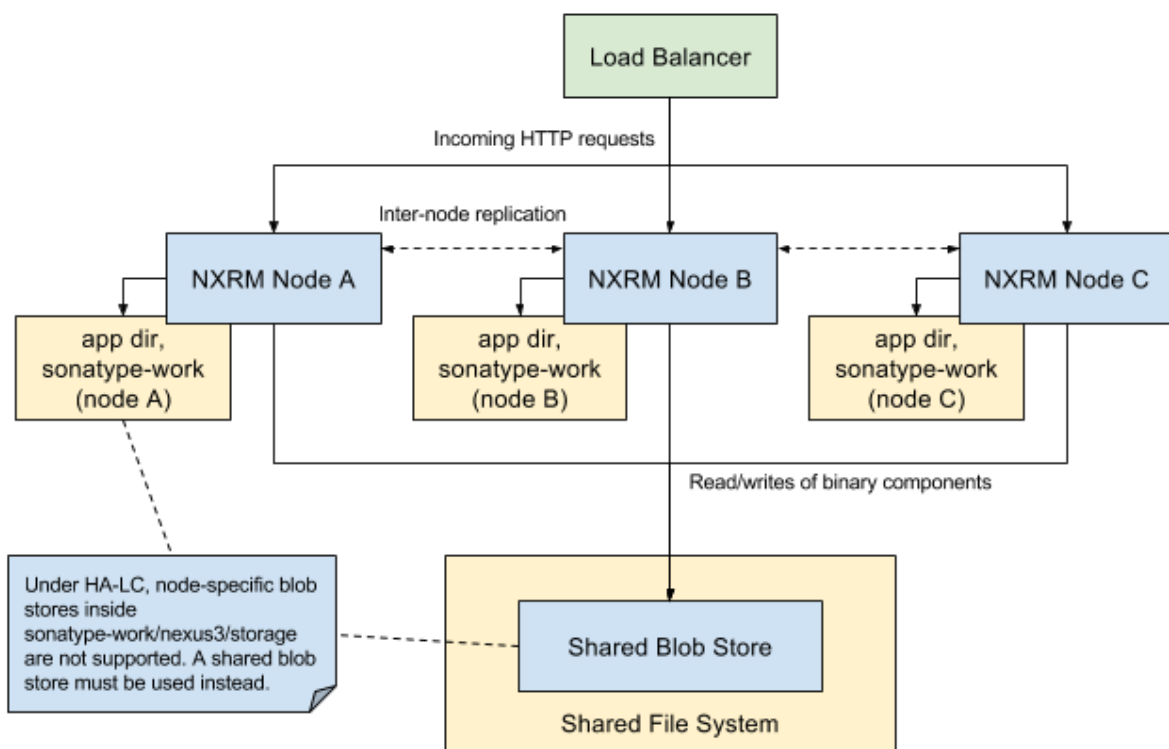
High-level Overview.....	16
Backing up Shared File Systems for your Blob Store(s).....	17
Generic NFS.....	17
NFS over Amazon Elastic File System (EFS).....	17
Amazon Simple Storage Service (S3).....	17
Backing up local storage for your Nodes	17
Restoring from backup	19
Initial Setup - High Availability.....	20
Node Deployment Steps for High Availability	20
First Node	20
Second Node and Beyond	21
Enabling High Availability on an existing Nexus Repository Manager Deployment	22
Blob Store Content Migration.....	23
Configuration.....	23
Operating your cluster	23
Startup and Confirming Node Connectivity.....	23
Running in Docker	24
Shutdown.....	24
Backup	24
Testing your Backup/Restore process	25
Node Names	25
Maintaining log files	25
Verifying Synchronization.....	26
Database Quorum and Cluster Reset	26
Troubleshooting.....	27
Monitoring Node Health	28
Task Management	28
Admin - Execute script.....	28
Export Configuration & Metadata for Backup Task	29
Single-node Tasks.....	29
Multi-node Tasks	29
JMX Lifecycle Operations	29

ManagedLifecycleBean	30
JMX Maintenance Operations	32
DatabaseMaintenanceBean	33
Upgrading your cluster	37
Upgrading an HA-C Environment from 3.x to 3.y	37
Rollback a Failed HA-C Upgrade	38
Load Balancing	38
Overview	39
Ports	39
Docker Repositories.....	39
Sticky Sessions	39
Monitoring Node Health	39
SSL Termination.....	40

 Available in Nexus Repository Pro only

Overview

High Availability Clustering (HA-C) is a feature designed to improve uptime by having a cluster of redundant NXRM "nodes" (instances) within a single data center. This feature allows you to maintain availability to your Nexus Repository Manager in the event one of the nodes becomes unavailable. A typical NXRM HA-C cluster is shown below:



Software and Hardware Requirements

Here's a list of the most important things you will need:

- A **test environment** to evaluate your cluster configuration and processes
- Nexus Repository Manager **Professional license** and **the latest version**

- A **load balancer**, such as [NGINX](#)¹ or [Apache HTTP](#)² or [AWS ELB](#)³
- **3 separate instances** of NXRM installed on **3 different systems** (e.g. 3 different EC2 instances) in a single datacenter
- **Network connectivity between the 3 different systems** so the NXRM can communicate with each other on several ports
- **Separate *local* and *shared* storage** (the difference will be described below)
- Each node needs to be configured according to our [general requirements](#)⁴.

In addition to these items, this guide will clarify the key concepts of **Node**, **Blob Store**, and **Hazelcast** and outline the operational considerations specific to HA-C.

Known Issues and Limitations

- The initial functionality intends to support a three-node cluster in a **single datacenter**. Cross-datacenter (WAN) replication is not supported at this time. For AWS, all nodes must be in the same availability zone.
- [Upgrading](#)(see [page 37](#)) an already established cluster to a new version of NXRM requires shutting down all nodes. Support for rolling upgrades is not yet available.
- Apt, Cocoapods, Conan, Conda, Go, Helm, p2 and R formats are currently disabled.

Topics in this section

- [Configuring Nodes](#)(see [page 6](#))
- [Configuring Blob Stores](#)(see [page 7](#))
- [Configuring Hazelcast](#)(see [page 11](#))
- [Designing your Cluster Backup/Restore Process](#)(see [page 16](#))
- [Initial Setup - High Availability](#)(see [page 20](#))
- [Operating your cluster](#)(see [page 23](#))
 - [JMX Lifecycle Operations](#)(see [page 29](#))
 - [JMX Maintenance Operations](#)(see [page 32](#))
 - [Upgrading your cluster](#)(see [page 37](#))
- [Load Balancing](#)(see [page 38](#))

¹ <https://www.nginx.com/>

² <https://httpd.apache.org/>

³ <https://aws.amazon.com/elasticloadbalancing/>

⁴ <https://help.sonatype.com/display/NXRM3/System+Requirements>

Configuring Nodes

What is a Node?

We refer to each separate instance of NXRM in a high availability cluster as a node. Nodes can run on physical or virtual servers, containers, or cloud services like Amazon Web Services EC2 instances. While multiple nodes can exist on the same physical or virtual server, each bound to different ports for testing purposes, but such a setup is discouraged in production as it does not provide for a redundant clustered environment.

Each node needs its own local storage for the `$data-dir` folder. This folder must not be shared by any other node in the cluster. Within the local storage, a node will store:

- local instance configuration (network ports, nexus.properties, logging)
- local log files
- a complete copy of the internally managed [OrientDB](#)⁵ databases NXRM uses to store configuration and asset metadata
- a complete copy of the [ElasticSearch](#)⁶ indexes

In addition, all nodes need shared storage for [blob stores](#)(see page 7).

How many Nodes will you need?

NXRM HA-C is intended for use with 3 nodes. It will not work properly with less than 3.

Any node can accept a write, but for the write to be committed it must be replicated to a 'majority' of available running nodes. [OrientDB](#)⁷ calculates a majority using the following formula: $(\text{number of expected running nodes} / 2) + 1$

A write (e.g. publishing assets, changing system configuration) will fail if nodes in your cluster unexpectedly lose connectivity or die and fewer than the calculated majority are running. However you can still write to your NXRM HA-C environment with only 1 running node as long as the other members of the cluster have either a) not started yet or b) were shutdown cleanly. The NXRM HA-C feature includes [monitoring](#)⁸ for this condition and controls to help you get your cluster running again.

⁵ <http://orientdb.com/orientdb/>

⁶ <https://www.elastic.co/products/elasticsearch>

⁷ <https://orientdb.com/docs/2.2/Distributed-Configuration.html#default-configuration>

⁸ <https://help.sonatype.com/display/NXRM3M/Operating+your+cluster#Operatingyourcluster-MonitoringNodeHealth>

⚠ Adding more than 3 nodes will result decreased performance due to the additional overhead incurred by database synchronization between the nodes. We recommend using no more and no less than 3 nodes in an HA cluster.

What data is replicated between Nodes?

NXRM HA-C uses [Hazelcast](https://hazelcast.org/)⁹ to keep the OrientDB databases completely in sync in an "Active-Active" configuration.

The ElasticSearch indices on the running nodes operate independently and do not share data; they do automatically index all content replicated in OrientDB.

Blobs, the assets (.jar, .xml, .tar.gz, docker images, etc.) in your repositories, are not replicated. They are stored in your blob stores.

ℹ Your next step is to plan your blob stores with [Configuring Blob Stores](#)(see page 7) before proceeding to [Initial Setup - High Availability](#)(see page 20).

Configuring Blob Stores

What is a Blob Store?

The binary assets you download via proxy repositories, or publish to hosted repositories, are stored in the blob store attached to those repositories. In traditional, single node NXRM deployments, blob stores are typically associated with a local filesystem directory, usually within the `sonatype-work` directory.

For NXRM HA-C however, the blob store location must be:

1. outside of the `sonatype-work` directory
2. read-write accessible by all nodes

ℹ Caution: Sonatype has very specific guidelines on choosing an appropriate filesystem for blob stores in a NXRM HA-C environment.

⁹ <https://hazelcast.org/>

Choosing a Type for your Blob Stores

NXRM HA-C stores blobs in shared storage, which can be either a shared file system or a cloud object store. Here are some recommendations to consider when choosing where to store your blobs. In general, file system blob stores may have better performance, but object store blob stores offer unbounded storage and convenience. Cloud object stores perform better if you are also running NXRM in the cloud with the same cloud provider.

❗ Do not copy or share the `sonatype-work` directory used by your Nexus Repository Manager nodes. Each node must be allowed to initialize its own private `sonatype-work` directory; data generated on first run is unique to each instance.

Your shared file systems, if File based, must exist outside of the `sonatype-work` directory.

Recommended Shared File Systems

NFS v4 is the recommended file system. On [Amazon Web Services](#)¹⁰ (AWS), [Elastic Filesystem](#)¹¹ (EFS) can be used as an NFS v4 server. Whatever the file system type, the blob store location must be outside of the `sonatype-work` directory and read/write accessible by all nodes.

NFS v4

The recommended settings for an NFS v4 server in `/etc/exports`:

```
/srv/blobstores 10.0.0.0/8(rw,no_subtree_check)
```

The recommended settings for mounting the NFS filesystem on each node:

```
defaults,noatime,rsize=1048576,wsiz=1048576,hard,timeo=600,retrans=2,vers=4.1
```

⚠ Versions of NFS older than v4 are not supported.

¹⁰ <https://aws.amazon.com/>

¹¹ <https://aws.amazon.com/efs/>

AWS Elastic Filesystem (EFS)

EFS acts as a limitless NFS v4 server and is an option if NXRM is running in AWS. Clients should mount the EFS volume with the same settings as NFS v4. Note that EFS performance will be lower than a dedicated NFS server.

Cloud Object Stores

Cloud object stores store your blobs using AWS S3. No shared file system is required if cloud object stores are used exclusively.

AWS Simple Storage Service (S3)

Blob stores can be configured to use [AWS Simple Storage Service](#)¹².

Blobs are stored in an S3 bucket by using AWS REST APIs over HTTP. Object PUTs use multi-part uploads of approximately 5MB chunks. Since HTTP traffic to S3 blobstores introduces more I/O latency across the network, **it is recommended to use S3 only if NXRM is running on EC2 instances within AWS**

When setting up a bucket for the S3 blob store:

- NXRM will automatically apply a lifecycle rule to expire deleted content.
- The bucket can use server-side encryption with KMS key management transparently. Other methods of server side encryption are not supported.

The AWS user for accessing the S3 Blobstore bucket needs to be granted permission for these actions:

- s3:PutObject
- s3:GetObject
- s3:DeleteObject
- s3:ListBucket
- s3:GetLifecycleConfiguration
- s3:PutLifecycleConfiguration
- s3:PutObjectTagging
- s3:GetObjectTagging
- s3:DeleteObjectTagging
- s3:DeleteBucket
- s3>CreateBucket
- s3:GetBucketAcl ([used for problem diagnosis](#)¹³) **NEW IN 3.19.0**

¹² <https://aws.amazon.com/s3/>

¹³ <https://issues.sonatype.org/browse/NEXUS-19494>

Sample minimal policy where <user-arn> is the ARN of the AWS user and <s3-bucket-name> the S3 bucket name:

```
{
  "Version": "2012-10-17",
  "Id": "NexusS3BlobStorePolicy",
  "Statement": [
    {
      "Sid": "NexusS3BlobStoreAccess",
      "Effect": "Allow",
      "Principal": {
        "AWS": "<user-arn>"
      },
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:ListBucket",
        "s3:GetLifecycleConfiguration",
        "s3:PutLifecycleConfiguration",
        "s3:PutObjectTagging",
        "s3:GetObjectTagging",
        "s3:DeleteObjectTagging",
        "s3:DeleteBucket",
        "s3:CreateBucket",
        "s3:GetBucketAcl"
      ],
      "Resource": [
        "arn:aws:s3:::<s3-bucket-name>",
        "arn:aws:s3:::<s3-bucket-name>/*"
      ]
    }
  ]
}
```

Access Denied Errors

Access denied errors are an indication that the user configured to connect to the bucket has insufficient permissions. AWS does not provide information specific to which permission is required to perform the failed action. For this reason NXRM will generically report access denied errors.

File Systems to avoid

Using the File Blob Store with NXRM HA-C relies on POSIX semantics. File systems known to be unreliable are:

1. glusterfs
2. FUSE based user space filesystems

Determining a Backup Solution

Backing up your blob store file systems is covered in detail in the [Designing your Cluster Backup/Restore Process](#)(see page 16) section.

i Your next step is to review your network configuration with [Configuring Hazelcast](#)(see page 11).

Configuring Hazelcast

Network Preparation

The nodes in a Nexus Repository Manager cluster need to be able to communicate with each other over TCP/IP. For cluster communications, Nexus Repository Manager will use a range of ports starting with 5701. Each node in the Nexus Repository Manager cluster will require that an extra port is available; the extra ports used by Nexus Repository Manager will be bound sequentially by default.

For example, in a three-node cluster, each of the nodes in the cluster will need to have ingress opened on ports

5701

, 5702, and 5703. The nodes will use ephemeral ports, randomly selected by the operating system, for outbound (egress) communications. If outbound or inbound network communications need to be customized and may be blocked by a firewall or other network appliance, the ports used for cluster communications can be customized in the Hazelcast configuration.

i For more information on customizing Hazelcast and the ports it uses, please see the documentation for [Hazelcast cluster configuration](#)¹⁴.

The nodes in a Nexus Repository Manager cluster will also replicate database transactions among the nodes in the cluster. The database requires ports 2424 and 2434 be open for ingress and egress to each of the rest of the nodes in the cluster.

¹⁴ <http://docs.hazelcast.org/docs/3.6/manual/html-single/index.html#setting-up-clusters>

Node Discovery

Hazelcast has multiple methods for discovering other nodes. In the default configuration, Nexus Repository Manager uses multicast to discover other Nexus Repository Manager nodes. This is done to simplify cluster configuration. However, multicast may not work reliably in all network environments.

To customize discovery, copy `NEXUS_HOME/etc/fabric/hazelcast-network-default.xml` to `$data-dir/etc/fabric/hazelcast-network.xml` and adjust the settings enclosed in the `<join>` tag.

❗ In NXRM 3.6.1 or earlier, these changes must be applied directly to `NEXUS_HOME/etc/fabric/hazelcast.xml`, where `NEXUS_HOME` is where Nexus Repository Manager is installed.

Multicast Discovery

Multicast is the default discovery method and is recommended unless it is not supported on your network. To test multicast connectivity between nodes, we recommend using [iPerf](#)¹⁵. iPerf is freely available for Windows, Linux and Mac OSX under the BSD licence.

To test multicast connectivity between two nodes, each node will need to have iPerf installed (note: iPerf3 does not support multicast client testing). During testing, one node will act as the "server" while the other node acts as the "client." This is important during testing, and further, we suggest that each node be tested as a server and as a client to ensure proper two-way multicast communication.

Testing multicast from the server side

```
iperf -s -B 224.2.2.3 -p 54327 -i 1
```

Testing multicast from the client side

```
iperf -c 224.2.2.3 -p 54327 -u -i 1
```

The address and port `224.2.2.3:54327` was chosen because this is the default port and address that will be used by Nexus Repository Manager during the node discovery. When the client is able to successfully connect and communicate with the server node, the client will output a set of brief diagnostic messages indicating how much data was sent and received. The following is a sample of the output from a client in a successful session:

¹⁵ <https://iperf.fr/>

Sample Output

```
Client connecting to 224.2.2.3, UDP port 54327
Sending 1470 byte datagrams
Setting multicast TTL to 1
UDP buffer size: 208 KByte (default)

[ 3] local 10.10.0.102 port 33743 connected with 224.2.2.3 port 54327
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0- 1.0 sec    129 KBytes  1.06 Mbits/sec
[ 3] 1.0- 2.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 2.0- 3.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 3.0- 4.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 4.0- 5.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 5.0- 6.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 6.0- 7.0 sec    129 KBytes  1.06 Mbits/sec
[ 3] 7.0- 8.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 8.0- 9.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 9.0-10.0 sec    128 KBytes  1.05 Mbits/sec
[ 3] 0.0-10.0 sec    1.25 MBytes 1.05 Mbits/sec
[ 3] Sent 893 datagrams
```

The advantage of the using multicast for Nexus Repository Manager node discovery is that nodes can be added and removed from the cluster without cluster administrators needing to perform configuration or configuration changes. However, routers may not be able to route multicast requests properly between subnets, or multicast may be disabled altogether. In these situations the cluster configuration can be done manually. If multicast is not available, either [AWS Discovery](#)(see page 13) (if you are running NXRM inside AWS on EC2 instances) or [TCP/IP Discovery](#)(see page 15) can be used.

For a production instance, we advise that the default address and port (224.2.2.3:54327) is not used, to avoid another cluster within the same network and using the same default configuration from unintentionally joining an existing cluster.

AWS Discovery

Nexus Repository Manager can be deployed on cloud-computing services, such as Amazon Web Services (AWS), where multicast discovery is not supported. Hazelcast AWS discovery is recommended.

AWS Environment

The NXRM servers need permissions to find other nodes, granted through [AWS Identity and Access Management](#)¹⁶ (IAM). The following configuration settings should be used:

1. The EC2 instances running NXRM should be assigned an InstanceProfile

¹⁶ <https://aws.amazon.com/iam/>

2. The InstanceProfile should have an InstanceRole with a policy granting the permission `ec2:DescribeInstances` on all resources.

In CloudFormation, the role would look similar to this:

```
"InstanceRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [{
        "Effect": "Allow",
        "Principal": {
          "Service": "ec2.amazonaws.com"
        },
        "Action": [
          "sts:AssumeRole"
        ]
      }]
    },
    "Policies": [{
      "PolicyName": "hazelcastDiscovery",
      "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Action": [
            "ec2:DescribeInstances"
          ],
          "Resource": ["*"]
        }]
      }
    }]
  }
}
```

NXRM Configuration for AWS Discovery

To configure Hazelcast for automatic node discovery, you need the IAM role name, AWS region, and security group of the EC2 instances. Find the `<join>` tag in `$install-dir/etc/fabric/hazelcast-network.xml`. Then, edit the file for each node:

1. Change the value in `<multicast enabled="true">` to `"false"`.
2. Update the `<discovery-strategies>` section as explained below.
3. Save the file.
4. Reboot each node in the cluster.

The `$data-dir/etc/fabric/hazelcast-network.xml` file with the modified properties will look similar to this:

```

<join>
  <!-- deactivating other discoveries -->
  <multicast enabled="false"/>
  <tcp-ip enabled="false"/>
  <aws enabled="false"/>

  <discovery-strategies>
    <discovery-strategy enabled="true" class="com.hazelcast.aws.AwsDiscoveryStrategy">
      <properties>
        <!--
name      | Required: use the command 'aws iam list-instance-profiles' on the EC2 instance to locate the
          | of the role used by the IAM Instance Profile you created previously
        -->
        <property name="iam-role">EC2_IAM_ROLE_NAME</property>


        <!-- Required: set this to the region where your EC2 instances running Nexus Repository Manager
are located -->
        <property name="region">us-west-1</property>

        <!--
hosts     | The next few options give you a choice to how the nodes are enumerated. You can specify:
          | * Just a security group name, if the security group only contains Nexus Repository Manager
          | * or a tag-key/tag-value pair, if you have tagged the EC2 instances
          | * or a combination of the two, if the security group has other hosts in it.
        -->
        <property name="security-group-name">EC2_SECURITY_GROUP_NAME</property>

        <!-- example tag idea, you are free to use whatever tag naming convention you need-->
        <property name="tag-key">Purpose</property>
        <property name="tag-value">Nexus Repository Manager</property>

      </properties>
    </discovery-strategy>
  </discovery-strategies>
</join>

```

 For more information on configuring Hazelcast in an AWS environment, please see the documentation for [the AWS EC2 discovery plugin for hazelcast](#)¹⁷.

TCP/IP Discovery

When multicast discovery and AWS discovery are not available, TCP/IP discovery can be configured. TCP/IP discovery requires the IP address of each node to be included in the configuration. Please see the

¹⁷ <https://github.com/hazelcast/hazelcast-aws>

documentation for [Hazelcast cluster configuration](#)¹⁸. Find the `<join>` tag in `$install-dir/etc/fabric/hazelcast-network.xml`. Then, edit the file for each node:

1. Add or set the following in `$data-dir/etc/nexus.properties`: `nexus.hazelcast.discovery.isEnabled=false`
2. Update the `<join>` section as explained below.
3. Save the file.
4. Reboot each node in the cluster.

In this case the `$data-dir/etc/fabric/hazelcast-network.xml` file with the modified properties will look similar to this:

```
<join>
  <multicast enabled="false"/>
  <tcp-ip enabled="true">
    <member-list>
      <member>10.0.1.10</member>
      <member>10.0.1.11</member>
      <member>10.0.1.12</member>
    </member-list>
  </tcp-ip>
  <aws enabled="false"/>
</join>
```

Designing your Cluster Backup/Restore Process

You must have a reliable, tested process for backing up your Nexus Repository Manager deployment .

This section will cover what you need to create a backup and restore process appropriate for your High Availability deployment .

High-level Overview

An appropriate backup process for Nexus Repository Manager needs to capture state from two separate, but interdependent parts of your deployment:

- the contents of each blob store
- a copy of the local storage for each node (i.e. the `$data-dir` directory, including the OrientDB databases within)

In the worst case scenario, a Nexus Repository Manager High Availability environment can be recreated minimally from:

¹⁸ <http://docs.hazelcast.org/docs/3.6/manual/html-single/index.html#setting-up-clusters>

- the contents of each blob store
- a complete set of OrientDB database exports from one of the nodes participating in the cluster

Backing up Shared File Systems for your Blob Store(s)

i Nexus Repository Manager requires that you choose and execute a separate backup process appropriate for the file systems under your blob store(s).

Generic NFS

Tools that back up the file system under the blob store can be safely executed while Nexus Repository Manager is running.

NFS over Amazon Elastic File System (EFS)

If you are using file backed blob stores on top of Amazon Elastic File System, you will need to review [Amazon's documentation specific to backing up EFS](#)¹⁹.

Amazon Simple Storage Service (S3)

The backup process for your S3 backed blob stores will involve creating a separate S3 "bucket" and periodically synchronizing the content from the source bucket under your blob store(s).

The [AWS Command Line Interface \(CLI\)](#)²⁰ provides an `s3 sync` command that you can invoke periodically to perform this:

- <http://docs.aws.amazon.com/cli/latest/reference/s3/sync.html>

There are also a number of third party tools that can perform this task.

Backing up local storage for your Nodes

Each node in your Nexus Repository Manager High Availability cluster has its own separate `$data-dir` directory. The `$data-dir/db` directory within contains OrientDB databases and requires special treatment.

¹⁹ <http://docs.aws.amazon.com/efs/latest/ug/efs-backup.html>

²⁰ <https://aws.amazon.com/cli/>


The `db` directory cannot be backed up by basic file system backup tools while Nexus Repository Manager is running. Instead create an *Admin - Export databases for backup* task as described in the [Backup and Restore](#)²¹ section.

Please note:

- All nodes in your Nexus Repository Manager cluster will automatically enter Read-Only mode when the backup task is running on any of the nodes
- It is recommended to schedule the backup task on at least 2 nodes. The tasks should run at different times to avoid concurrency issues. The backup from any node can be used to perform a restore, but running on more than one node ensures that there isn't a single point of failure for the backup.
- The folder specified in the backup task for a node must be available and writable by the user running Nexus Repository Manager on the nodes
- The backup task produces a set of files each time it is run that must be protected. Be sure to use a file system backup tool to store these files separately.

For the rest of the content within the `$data-dir` directory, traditional file system backup tools will suffice. It is safe to skip backing up the following directories within `$data-dir`:

- `backup`
- `cache`
- `db`
(backed up via the *Admin - Export databases for backup* task)
- `elasticsearch` (generated via the *Repair - Rebuild repository search* task)
- `logs` (see the [Operating your cluster](#)(see [page 23](#)) section for preserving logs)
- `tmp`

 We recommend you execute backup of your local storage at the same starting time as your blob store file system backup.

²¹ <https://help.sonatype.com/display/NXRM3/Backup+and+Restore>

Restoring from backup

❗ If you determine you need to restore your Nexus Repository Manager deployment from backup, you must first focus only on starting one node and stabilizing it before you add new nodes to the High Availability cluster.

Start this process by ensuring all Nexus Repository Manager nodes in your cluster are offline.

Assuming all nodes in your cluster are offline:

1. Identify the host that will be the first node of your restored deployment. Locate the Orient DB exports from that node and identify the date and time they were taken. Use this same (or as close to) date and time when restoring the blob store file systems.
2. Restore the blob store file systems and host connectivity. Be sure to restore the blob store file system to the exact same absolute file system path used previously; don't try to restore to a different file system path.
3. Restore the local storage (`$data-dir`) for the node. Again, preserve the same absolute file system paths used previously.
4. Remove the following directories from `$data-dir/db`
 - `component`
 - `config`
 - `security`
5. Copy the complete set of OrientDB exports from that node to `$data-dir/restore-from-backup` for restoration (**Note:** For version 3.10.0 or earlier use `$data-dir/backup` as the restore location).
6. Start Nexus Repository Manager on only this node.

Confirm that your instance is in a stable state before proceeding:

- Inspect configuration and confirm it matches expectations
- Attempt to retrieve and/or publish new components

Once the first node is online and stable, it is safe to proceed in bringing additional nodes back into your cluster. Focus on adding only one node at a time to the cluster. On each of the additional nodes, delete the following folders from within `$data-dir/db` prior to starting Nexus Repository Manager

- `component`
- `config`
- `security`

The OrientDB databases removed from the additional node will automatically be rebuilt from the instances already running in the High Availability cluster.

Initial Setup - High Availability

This section will cover the steps for enabling High Availability in your Nexus Repository Manager deployment.

⚠ This document assumes that you have completely read and prepared your choices for [Configuring Blob Stores](#)(see page 7), [Configuring Hazelcast](#)(see page 11), and [Designing your Cluster Backup/Restore Process](#)(see page 16).

Do not proceed until you have a complete plan ready for all 3.

Once you begin, focus on stabilizing one single node at a time.

Node Deployment Steps for High Availability

First Node

1. Follow the usual [Installation Methods](#)²². Create the file `sonatype-work/nexus3/etc/nexus.properties` with the following contents:

```
# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-args=${jetty.etc}/jetty.xml,$jetty.etc/jetty-http.xml,$jetty.etc/jetty-requestlog.xml
nexus-context-path=/

# Nexus section
nexus-edition=nexus-pro-edition
nexus-features=\
  nexus-pro-feature

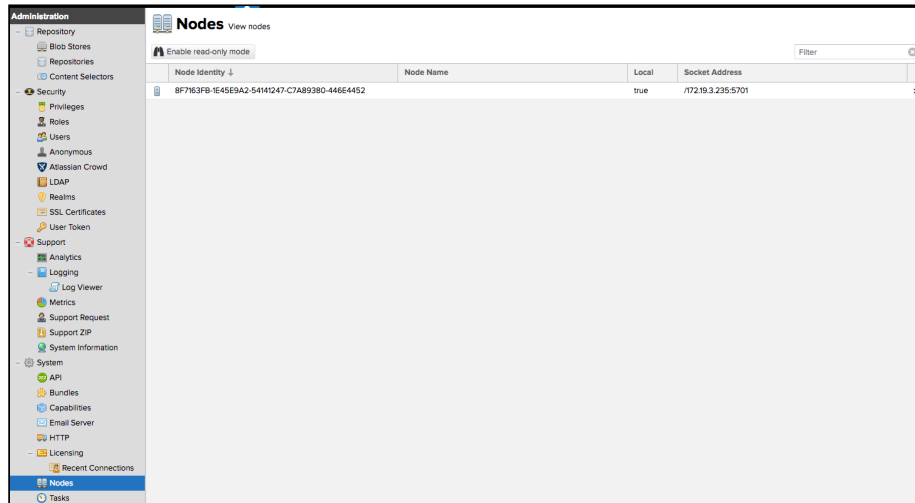
nexus.clustered=true
# nexus.licenseFile is only necessary for the first run
# replace /path/to/your/sonatype-license.lic with the path to your license, and ensure the user
# running Nexus Repository manager can read it
nexus.licenseFile=/path/to/your/sonatype-license.lic
```

2. Next, deploy your chosen [hazelcast configuration](#)(see page 11) to the path `sonatype-work/nexus3/etc/fabric/hazelcast-network.xml`. For NXRM 3.6.1 or earlier, these changes must be made

²² <https://help.sonatype.com/display/NXRM3/Installation+Methods>

in `NEXUS_HOME/etc/fabric/hazelcast.xml`, where `NEXUS_HOME` is where Nexus Repository Manager is installed.

3. Start Nexus Repository Manager. Log in as an administrator, then visit the *System* → *Nodes* screen. You should see your first node, like this example:



You may want to assign a friendly *Node Name* to each node in your cluster to make it easier to identify. Without a node name, screens that refer to individual nodes will display the unique ID labeled *Node Identity* in the screenshot above. Simply click on the entry in the list and you will have the ability to set a node name.

When Nexus Repository Manager is started with `nexus.clustered=true` and a PRO license, it will not create the default blobstore or initial example repositories. You are free to set these up now, or later as you add nodes.

Second Node and Beyond

⚠ Do not copy the `sonatype-work` directory to your additional nodes. Each node must be allowed to initialize its own private `sonatype-work` directory; data generated on first run is unique to each instance.

Starting a second node off of a copy of the `sonatype-work` directory will result in the inability to correctly form a cluster.

Again follow the usual [Installation Methods](https://help.sonatype.com/display/NXRM3/Installation+Methods)²³. Before starting Nexus Repository Manager, repeat the steps you performed on the first node:

²³ <https://help.sonatype.com/display/NXRM3/Installation+Methods>

1. Create the file `sonatype-work/nexus3/etc/nexus.properties` with the same contents as the first node (note that you will have to choose a different port number only if you're running two or more nodes on a single host):

```
# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-args=${jetty.etc}/jetty.xml,${jetty.etc}/jetty-http.xml,${jetty.etc}/jetty-requestlog.xml
nexus-context-path=/

# Nexus section
nexus-edition=nexus-pro-edition
nexus-features=\
  nexus-pro-feature

nexus.clustered=true
# nexus.licenseFile is only necessary for the first run
# replace /path/to/your/sonatype-license.lic with the path to your license, and ensure the user
# running Nexus Repository manager can read it
nexus.licenseFile=/path/to/your/sonatype-license.lic
```

2. Deploy your chosen hazelcast configuration to the path `sonatype-work/nexus3/etc/fabric/hazelcast-network.xml`
3. Start Nexus Repository Manager

After each node joins the cluster, confirm it is visible in the *System* → *Nodes* screen. Set a node name for each node as desired. Repeat this section for each node you wish to join your High Availability Cluster until all are running.

Enabling High Availability on an existing Nexus Repository Manager Deployment

If you already have a single node Nexus Repository Manager deployment, you will still need to read and prepare your choices for [Configuring Blob Stores](#)(see page 7), [Configuring Hazelcast](#)(see page 11), and [Designing your Cluster Backup/Restore Process](#)(see page 16). You will also need to upgrade your single node deployment to a version of Nexus Repository Manager that supports [High Availability](#)(see page 4) before you attempt to establish a cluster. All nodes within a cluster must be running the exact same version of Nexus Repository Manager.

You additionally may have to design a strategy for migrating the content on your existing blob stores to a shared filesystem accessible to all nodes in the cluster. Moving blobs from one blobstore to another can be achieved using Dynamic Storage.

Blob Store Content Migration

All blob stores used by HA-C nodes must be shared. If you need to move blobstore blobs to new blobstores in preparation for sharing between nodes, first perform that procedure in a single node configuration using the [Dynamic Storage](#)²⁴ operations.

Configuration

Once you have all the pieces in place, enable high availability by performing the steps listed in the Second Node and Beyond section above.

Operating your cluster

! It is recommended that the nodes are started in the reverse order in which they were shutdown wherever possible (for example, if they were shutdown in the order A, B, C, then C should be started first, followed by B and then A). The last node shutdown will have the most current data and should be the first one started up after a full cluster shutdown.

Startup and Confirming Node Connectivity

After enabling HA-C for your Nexus Repository Manager installation, check the console or log file after launching a clustered node to confirm that all nodes have been detected.

When you start the nodes, you will see a message in the nexus.log confirming the connection of the cluster members, like the one below:

```
2017-11-06 11:04:12,045-0500 INFO [hz.nexus.generic-operation.thread-0] *SYSTEM
com.hazelcast.internal.cluster.ClusterService - [172.19.3.78]:5703 [nexus] [3.7.8]

Members [3] {
  Member [172.19.3.78]:5701 - 6f4df1bc-606d-4821-8a3b-0980f093abd1
  Member [172.19.3.78]:5702 - 436158e4-2865-40fe-bfaa-d350db8dedb1
  Member [172.19.3.78]:5703 - 7ff93c39-23b3-4bf5-9b8d-51264ccecee2 this
}
```

²⁴ <https://guides.sonatype.com/repo3/technical-guides/scaling-dynamic-storage/>

Running in Docker

Running your Nexus Repository Manager cluster within a container network is supported. However, there is an important consideration. When running Nexus Repository Manager inside of a docker container, the default behavior of the "docker stop" command is to first send a TERM signal, but if the process has not quit within 10 seconds, a KILL signal is sent. This has been known to cause Nexus Repository Manager clusters to become corrupted.

WARNING


When running Nexus Repository Manager nodes inside docker containers, you will need to specify an extended timeout, such as

```
docker stop --time=120 <CONTAINER_NAME>
```

to allow nodes to exit the cluster cleanly.

Shutdown

A node leaves the cluster through a clean shutdown of Nexus Repository Manager. A proper shutdown is performed by using the `nexus.exe stop` command on Windows-based systems and `nexus stop` on all others. When the node is leaving by a clean shutdown, the node is unregistered from the cluster. If a node that is registered in the cluster becomes abruptly unavailable (for example, the node's network link is broken, or the node's operating system crashes), that node may remain registered in the cluster despite no longer participating in database replication.

 The clean shutdown is an important consideration when designing scripts for stopping a Nexus Repository Manager instance.

Nexus Repository Manager will initiate a shutdown if it receives the TERM signal, which is the default signal sent with the kill command on unix-like operating systems. A clean shutdown is also important for recovering from an outage event.


Backup

Make sure to take some time to read the section on [designing your backup and restore plan](#)(see page 16). Running Nexus Repository Manager in a cluster does not reduce the need for a robust backup strategy. With a cluster, you will need to specify the repository manager instance where the task will execute. This is

necessary because the backup will result in exported database backup files. These files will be placed in the configured location, on the chosen node's file system. However, since the task is relegated to a single node, this introduces the possibility that the database backup may not be executed if the node is no longer participating in the cluster. In some environments it may be preferable to configure two different nodes to execute backup tasks on alternating schedules. While the [Backup and Restore](#)²⁵ task is running, the Nexus Repository Manager cluster will be in read-only mode.

Testing your Backup/Restore process

We encourage you to test your [restore process](#)(see [page 19](#)) for disaster recovery minimally every 6 months. We also strongly suggest that you test the disaster recovery plan prior to upgrading Nexus Repository Manager to a new release.

 Please keep in mind Nexus Repository Manager requires that you choose and execute a separate backup process of your blob store(s) appropriate for the file system.

Node Names

When clustered, Nexus Repository Manager offers the ability to specify an alias for each node. Behind the scenes, Nexus Repository Manager generates a unique identifier for each node that joins a cluster. The alias that is specified by the administrator allows a logical mapping between the system-generated identifier and the specified name. The alias for a node is stored in `sonatype-work/nexus3/etc/node-name.properties`. The name can be updated in the Nexus Repository Manager user interface or directly in the properties file. If the name is updated directly in the properties file, it is necessary to restart the renamed node for the change to take effect. When an alias has been provided for a node, that alias will be used in place of the identifier where appropriate. For instance, the alias will be used in logging messages and in the user interface.

Maintaining log files

Clustered Nexus Repository Manager nodes will create the same set of log files as unclustered installations. However, in a cluster, the name of the node will be added to each log message. The name that is used in the log messages will be the given alias or default to the node's system-generated identifier. The log files on each node will be rotated daily. The log files for the previous day will be renamed and compressed to save disk space.

The Nexus Repository Manager user interface contains a log viewer available to administrators. This log viewer is limited to only display log messages that have been emitted on the local node. In an environment where traffic is routed through a load balancer, this view may not be helpful. In that case, we suggest that a


²⁵ <https://help.sonatype.com/display/NXRM3/Backup+and+Restore>

log aggregation system be put in place. Having a node identifier as a part of each log entry should enable better searching within such a system.

Verifying Synchronization

At run-time, the repository manager user interface allows you to view the status of the clustered nodes.

See [Nodes](#)²⁶ for details on viewing active nodes in a cluster.

 In the event a single node loses connection to the cluster, the remaining nodes will continue to make decisions on which data changes are valid. The disconnected node will reject further writes until it rejoins the cluster.

Database Quorum and Cluster Reset

To maintain data consistency, Nexus Repository Manager forces database transactions to be accepted and written to multiple nodes in the cluster. The goal is that all transactions are replicated within the cluster before they are accepted. However, this introduces some situations that need explanation. To allow for continued operation, even during some failures, the transactions are not expected to replicate to all nodes in a cluster before the transaction is accepted. The Nexus Repository Manager cluster does expect that the transaction is replicated to the majority of nodes within a cluster. This means that in a cluster with three nodes, no less than two nodes must accept a write before the transaction is considered a success. To calculate the size of the majority, Nexus Repository Manager will use the following equation:

 $\text{ROUNDED_DOWN}(\$N / 2) + 1$

Where $\$N$ represents the number of nodes registered in the cluster and `ROUNDED_DOWN` finds the largest whole number less than or equal to the given parameter.

Nodes register with the cluster when they are launched.

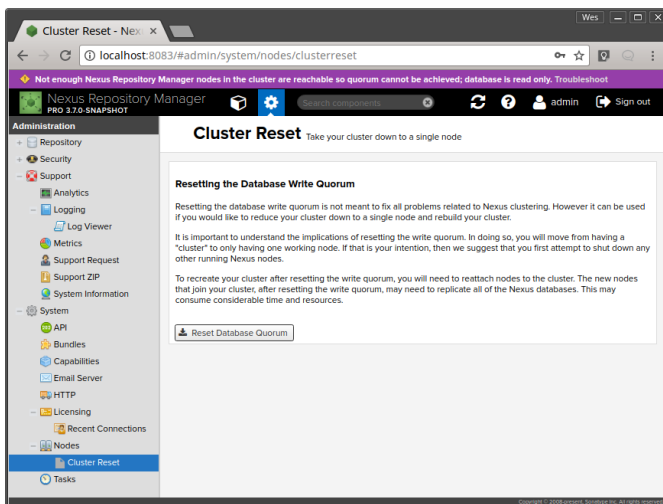
Although Nexus Repository Manager clustering replicates data across all nodes in the cluster, a cluster can be reduced to a single node. A single node cluster will accept write transactions. You can still write to your Nexus Repository Manager cluster environment with only one running node as long as the other members of the cluster have either been shutdown cleanly or they have not started yet. If nodes in your cluster unexpectedly lose connectivity or die and fewer than the calculated majority are running, then attempts to write (such as publishing assets, changing system configuration) will fail.

²⁶ <https://help.sonatype.com/display/NXRM3/System+Configuration#SystemConfiguration-Nodes>

Because of the quorum calculation and cluster registration algorithms, it is possible that Nexus Repository Manager's cluster table can enter a state where it does not accurately reflect the true state of your cluster. For example, if a registered member is abruptly removed from the cluster and a fresh installation is added to replace the failed node, then Repository Manager clustering may be calculating the write quorum for database transactions still considering the removed node. In this example, Nexus Repository Manager will continue to operate. However, after each abrupt removal of a node, it becomes more likely that a cluster could have difficulty achieving write quorum.

Troubleshooting

If your cluster enters a state where write quorum cannot be achieved, then Nexus Repository Manager will present the option to reset your cluster. The write quorum not reached warning appears as a horizontal bar across the top of the Nexus Repository Manager user interface, as shown in the screen capture below:



Before resetting your Nexus Repository Manager cluster, you should attempt to manually resolve the cluster issues. There are a few ways you can manually resolve the issues.

- Try to cleanly shutdown all clustered nodes, except one, and then try to bring your cluster back online by restarting the nodes one at a time waiting for each to finish starting before starting the next.
- You can also try to reconnect or restart nodes that may have been killed or disconnected. If the restarted nodes are not meant to be permanent members of the cluster, then performing a clean shutdown after restarting or reconnecting will properly remove the nodes from the cluster.

As a last resort, you can click the *Troubleshoot* link in the write quorum warning. On the cluster reset page, you will see the *Reset Database Quorum* button. When you click on the button, all entries are removed from the cluster table except for the node that processes the request to reset the cluster. In practice, this means that all other nodes need to be shutdown and removed from the load balancer rotation before resetting the cluster.

WARNING

It is only safe to reset the cluster once all other nodes have been cleanly shutdown.

After the reset completes, the cluster will immediately begin accepting writes on the remaining node. Nodes can then be added to the cluster and placed back into the load balancer rotation.

Monitoring Node Health

Once your HA environment is set up, you should monitor the health of the nodes in your cluster. The read-only status of the cluster is visible at the `http://<serveripaddress>:<port>/service/metrics/data` endpoint, as "readonly.enabled", under "gauges". See the [support article](#)²⁷ for the HTTP endpoints that HA-C exposes and/or the [support article](#)²⁸ for enabling JMX, for more information.

You can also use the load balancer friendly [status resource](#)²⁹ to check the health of the nodes in an HA cluster.

Task Management

In a Nexus Repository Manager cluster, it is necessary to consider which node or nodes execute a task. Knowing how the tasks run in a cluster will be an important element of monitoring cluster health. Nexus Repository Manager now places task logging output in a separate file that corresponds to the execution of the task. Many tasks will be executed on a random single node in the cluster to balance the resource usage across the cluster. Other tasks may execute simultaneously on all nodes in the cluster. Monitoring of the task must take into account the behavior of the execution of the task. The node requirements for common tasks are listed below.

Admin - Execute script

In a Nexus Repository Manager cluster, when you create an *Admin - Execute script* task, you are able to specify whether the task runs on all nodes simultaneously or whether the script is only executed on a single node. When a task manipulates a resource that is shared within the cluster, such as repository configuration or a blob store, the task should only be executed on a single node. However, if the task is interacting with a resource that is local to the node, such as the search indices, then the task needs to be configured with *Run task on all nodes in the cluster* selected.

²⁷ <https://support.sonatype.com/hc/en-us/articles/226254487>

²⁸ <https://support.sonatype.com/hc/en-us/articles/218501277>

²⁹ <https://help.sonatype.com/display/NXRM3M/Load+Balancing#LoadBalancing-MonitoringNodeHealthNEWINRELEASE3.15>

Export Configuration & Metadata for Backup Task

Please see the section on [designing your backup plan](#)(see page 16).

Single-node Tasks

The following Single-node Tasks will run on a single, randomly-selected node in a Nexus Repository Manager cluster:

- *Admin - Cleanup tags*
- *Admin - Compact blob store*
- *Admin - Delete orphaned API keys*
- *Admin - Log database table record counts*
- *Admin - Remove a member from a blob store group*
- *Docker - Delete incomplete uploads*
- *Docker - Delete unused manifests and images*
- *Maven - Delete SNAPSHOT*
- *Maven - Delete unused SNAPSHOT*
- *Maven - Publish Maven Indexer files*
- *Maven - Unpublish Maven Indexer files*
- *Repair - Rebuild Maven repository metadata (maven-metadata.xml)*
- *Repair - Rebuild Yum repository metadata (repodata)*
- *Repair - Reconcile component database from blob store*
- *Repository - Delete unused components*

The following Single-node Tasks will run on a single selected node in a Nexus Repository Manager cluster:

- *Admin - Export database for backup*
- *Repair - Rebuild repository browse*
- *Repair - Reconcile date metadata for blob store*

Multi-node Tasks

The following tasks may run on all nodes or on the current node (i.e. the node the UI is interacting with):

- *Admin - Execute script*
- *Repair - Rebuild repository search*
- *Repair - Reconcile npm /-/v1/search metadata*

JMX Lifecycle Operations

⚠ These operations are intended to be used with the guidance of Sonatype support. Usage without supervision of Sonatype is not supported or recommended.

ManagedLifecycleBean

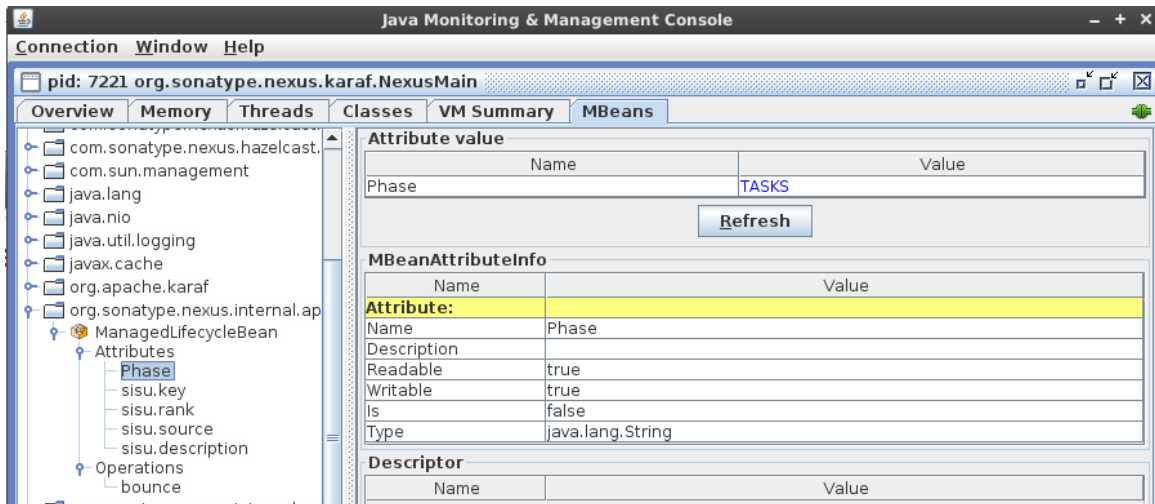
This JMX bean provides a set of attributes and operations to allow the user to inspect and manipulate the current lifecycle phase of a Nexus Repo Manager instance. The expectation is that these will be used for recovery operations. A Phase is a step in the start up process used to group similar components together and ensure that their dependencies are started before them. In order, the phases are: OFF, KERNEL, STORAGE, RESTORE, UPGRADE, SCHEMAS, EVENTS, SECURITY, SERVICES, CAPABILITIES, and TASKS.

1. OFF - NXRM is completely off.
2. KERNEL - The most basic parts of the node are running.
3. STORAGE - The databases and caches are setup.
4. RESTORE - Any restoring from backups happens in this phase.
5. UPGRADE - If any upgrades are needed, they will occur in this phase.
6. SCHEMAS - This creates any missing schemas in the database.
7. EVENTS - The framework that lets services communicate events is started.
8. SECURITY - The parts of NXRM responsible for security are enabled.
9. SERVICES - The bulk of an instance's components are started here.
10. CAPABILITIES - The capabilities are started, see [System Configuration](https://help.sonatype.com/display/NXRM3/System+Configuration#SystemConfiguration-AccessingandConfiguringCapabilities)³⁰
11. TASKS - The task scheduler is enabled, and tasks may run.

³⁰ <https://help.sonatype.com/display/NXRM3/System+Configuration#SystemConfiguration-AccessingandConfiguringCapabilities>

Attributes

Phase

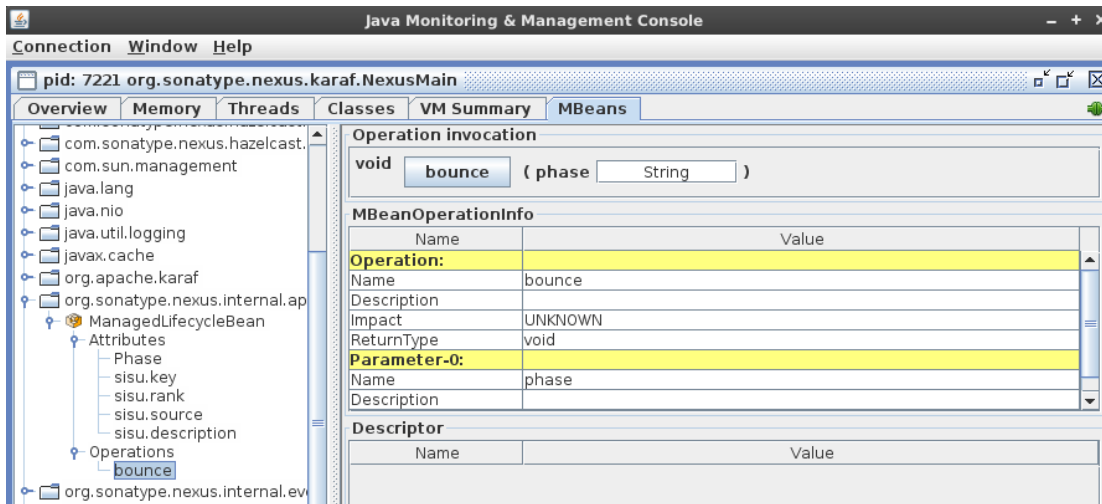


Possible Value: Case sensitive string, one of OFF, KERNEL, STORAGE, RESTORE, UPGRADE, SCHEMAS, EVENTS, SECURITY, SERVICES, CAPABILITIES, or TASKS.

This sets NXRM to run in a particular phase. By moving a Nexus Repo manager instance to a new phase, you can start or stop select components. For example, if your instance is running in the TASKS phase and you set the phase to CAPABILITIES, then that instance will not run any scheduled tasks until it is moved back to the TASKS phase.

Operations

Bounce



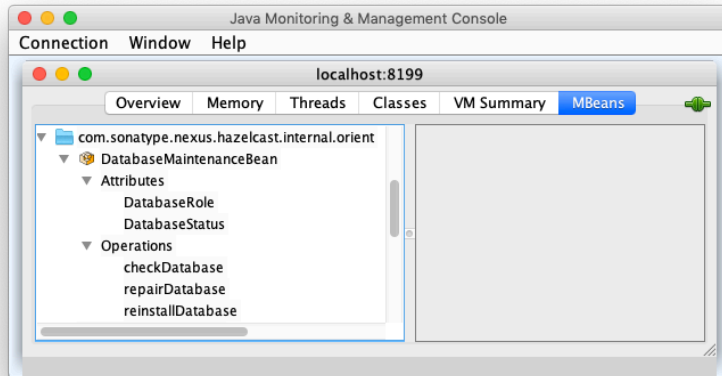
Possible Value for phase: Case sensitive string, one of OFF, KERNEL, STORAGE, RESTORE, UPGRADE, SCHEMAS, EVENTS, SECURITY, SERVICES, CAPABILITIES, or TASKS.

This operation takes the instance down to the supplied Lifecycle phase, and then back to the phase in which it was originally running. This effectively restarts the phases.

JMX Maintenance Operations

 **Available in Nexus Repository Pro**


DatabaseMaintenanceBean




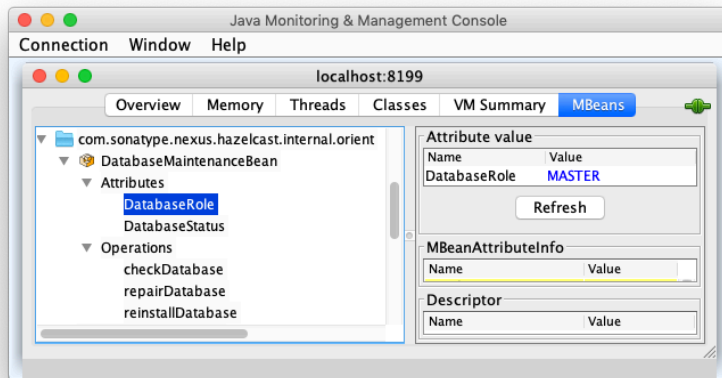
This JMX bean provides a set of attributes and operations to allow the user to inspect and manipulate the state of a node in Orient database cluster. The expectation is that these will be used mainly for troubleshooting and recovery operations.

Attributes

DatabaseRole

 Only available in HA-C

 Applies to all databases on the current node



Possible Values: MASTER, REPLICA

This attribute represents the role of the node in an Orient cluster. The role of MASTER represents a fully writable node in an Orient cluster which participates in the `writeQuorum`. The role of REPLICA represents a node in read-only mode which is accepting only idempotent commands, e.g. read and queries.

When setting a node's role to REPLICA an override is added to Orient's configuration so that node will always be a REPLICA, regardless of the cluster's default role. When the node is set back to MASTER that override is removed, so it will defer to the cluster's default role - which is MASTER unless it's been frozen.

References

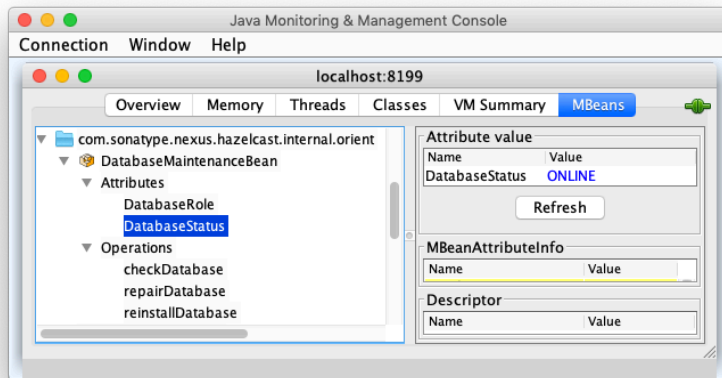
- [Distributed-Architecture: Server Roles](http://orientdb.com/docs/2.2.x/Distributed-Architecture.html#server-roles)³¹

DatabaseStatus

i Only available in HA-C


i Applies to all databases on the current node

³¹ <http://orientdb.com/docs/2.2.x/Distributed-Architecture.html#server-roles>



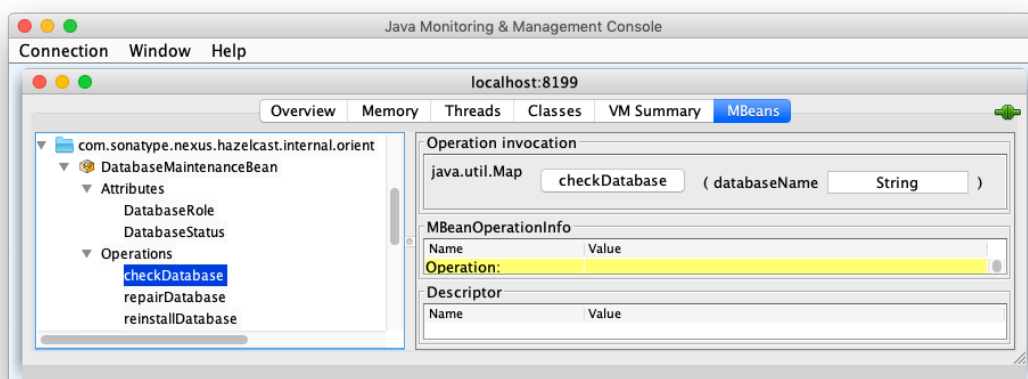
Possible Values: ONLINE, OFFLINE, NOT_AVAILABLE

This attribute represents the status of the node in an Orient cluster. The status of ONLINE represents a node in a normal state which is fully participating in the cluster. The status of OFFLINE represents a node that is not currently participating in the cluster. The status of NOT_AVAILABLE represents a node that is not currently available for some reason.

 Setting the database status to NOT_AVAILABLE triggers an automatic delta-sync with the rest of the cluster. If this is successful, it will move back to ONLINE.

Operations

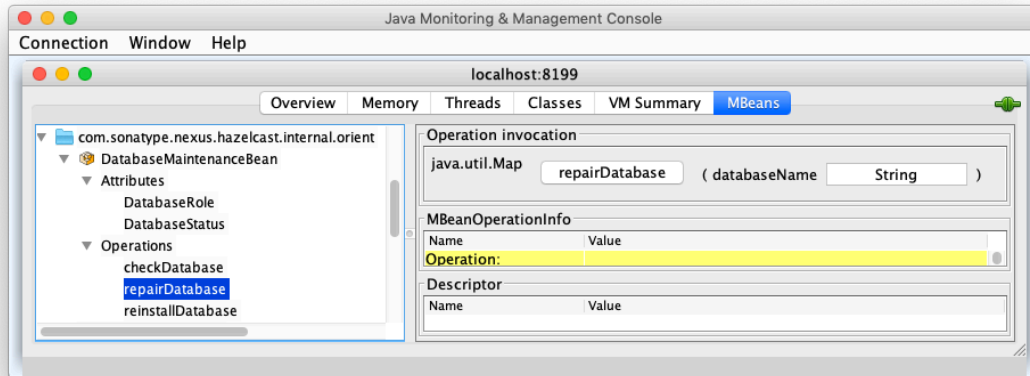
checkDatabase



Possible Database Names: accesslog, component, config, security

Checks database pages for corruption and checks that indices cover all records (i.e. no duplicates).


repairDatabase

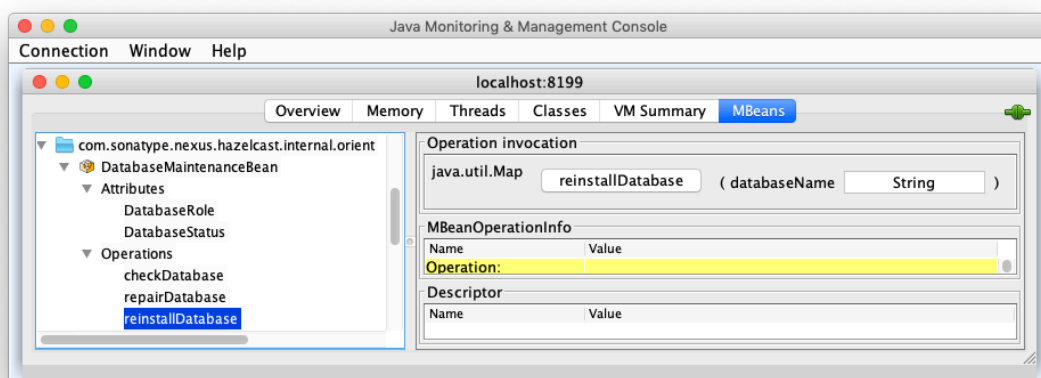


Possible Database Names: accesslog, component, config, security

Attempts to repair any corrupt database pages and rebuilds indices.


reinstallDatabase

 Only available in HA-C



Possible Database Names: accesslog, component, config, security

Attempts to reinstall the full database from the rest of the cluster.

 The user can't force a reinstall from a specific node, but typically Orient will choose the node that most recently took a backup or the oldest member (assuming that's not the node requesting the reinstall)

Upgrading your cluster

Upgrading an HA-C Environment from 3.x to 3.y

Prepare for HA-C Upgrade

Do not upgrade unless you have established a reliable and working [HA-C backup and restore](#)(see [page 19](#)) process. This includes a backup of shared blob stores filesystems.

HA-C **upgrade is not supported** and will fail when:

- clustering is enabled (`nexus.clustered=true`)
- databases are read-only
- restoring databases from a backup created with an earlier version

Download, unpack and [pre-configure new version distribution install files on each cluster node](#)³² before performing the upgrade.

Perform HA-C Upgrade

We will refer to your three nodes as **NODE-A**, **NODE-B** and **NODE-C**.

NODE-A will be upgraded to a new version in unclustered mode.

NODE-B and **NODE-C** will join and sync databases from **NODE-A** when they start for the first time with the new version.

For each individual node being upgraded, make sure you [adhere to the instructions for upgrading a single node](#)³³, especially the part of merging configuration file changes

1. Schedule an outage for your service; deny access to all nodes in the cluster using your load balancer.

³² <https://support.sonatype.com/hc/en-us/articles/115000350007-Upgrading-Nexus-Repository-Manager-3-1-0-and-Newer>

³³ <https://support.sonatype.com/hc/en-us/articles/115000350007-Upgrading-Nexus-Repository-Manager-3-1-0-and-Newer>

2. Perform an HA-C backup. [Backup the databases](#)(see page 16) of **NODE-A** and *at least one other node*.
3. Shut down **NODE-C** gracefully and wait for it to stop.
4. Shut down **NODE-B** gracefully and wait for it to stop.
5. Edit **NODE-A** `$data-dir/etc/nexus.properties` file and set `nexus.clustered=false` to disable clustering. Save the file.
6. Shut down **NODE-A** gracefully and wait for it to stop.
7. Start **NODE-A** using the new version. Wait for it to startup in un-clustered mode and perform automatic upgrade steps.
8. Verify **NODE-A** is functioning after the upgrade in un-clustered mode, e.g. use the UI and access repository content.
9. If **NODE-A** fails verification, follow our [HA-C upgrade rollback procedure](#)(see page 38).
10. If **NODE-A** passes verification, then delete the **NODE-B** and **NODE-C** `$data-dir/db` directory (you have already made database backups earlier).
11. Edit **NODE-A** `$data-dir/etc/nexus.properties` file and set `nexus.clustered=true` to enable clustering on the next startup. Save the file.
12. Restart **NODE-A**. Wait for it to startup in clustered mode.
13. Start **NODE-B** with the new version. Wait for it to startup in HA-C mode and sync the databases from **NODE-A**.
14. Start **NODE-C** with the new version. Wait for it to startup in HA-C mode and sync the databases from **NODE-A** or **NODE-B**.
15. Sign-in as an administrator and confirm that all nodes are visible in the *System* → *Nodes* administration page and that there are no status warnings about cluster health.
16. Verify that all nodes are functioning normally on the new version, e.g. use the UI and access repository content.
17. Restore access through your load balancer and resume operations.

Rollback a Failed HA-C Upgrade

Do not attempt to start a older version of NXRM against an already upgraded or partially upgraded [data directory](#)³⁴ or databases. This will fail.

The only supported way to rollback to an old version is to start NXRM against a set of databases and data directory were created using an identical NXRM version.

Load Balancing

³⁴ <https://help.sonatype.com/display/NXRM3/Directories>

Overview

Load balancing is required to achieve transparent redundancy in a High Availability Cluster. The following outlines considerations relating specifically to NXRM.

Ports

NXRM listens on port 8081 for HTTP by default. There is no default port for HTTPS. These ports are also configurable (see [NXRM Port Configuration](#)³⁵).

Docker Repositories

The docker client does not allow a context as part of the path to a registry, as the namespace and image name are embedded in the URLs it uses. NXRM allows custom ports to be exposed to create a listener on a root context. See [SSL and Repository Connector Configuration](#)³⁶ for more details. Alternatively you may choose to use a [URL rewriting scheme](#)³⁷ to achieve the same results without needing to configure custom ports (not covered here). Using URL rewriting allows you to sidestep this limitation, otherwise exposing and using the configured port(s) are required.

Sticky Sessions

The NXRM UI uses a cookie named NXSESSIONID to maintain a users state. The load balancer should be configured to enforce sticky sessions for requests containing this cookie.

Monitoring Node Health

NEW IN RELEASE 3.15.0

NXRM provides two endpoints you can use to check the status of your NXRM node. The *Readable Health Check* endpoint verifies that a node can handle read requests. The *Writable Health Check* endpoint verifies that a node can handle read and write requests. Since a node that can handle read requests but not write requests may still be considered serviceable, it is left to the user to decide which status endpoint to use.

These endpoints can be used by a load balancer to determine if a node should be considered to handle requests. For both status endpoints, Success is represented as HTTP 200 OK. Failure is represented as HTTP 503 SERVICE UNAVAILABLE. The nexus.log file for the node should be inspected for further details.

³⁵<https://help.sonatype.com/display/NXRM3/Configuring+the+Runtime+Environment#ConfiguringtheRuntimeEnvironment-ChangingtheHTTPPort>

³⁶ <https://help.sonatype.com/display/NXRM3/SSL+and+Repository+Connector+Configuration>

³⁷ <https://support.sonatype.com/hc/en-us/articles/360000761828>

The URLs for the Health Check endpoints are:

Name	URL
Readable Health Check	<code>http://<hostname>:<port>/service/rest/v1/status</code>
Writable Health Check NEW IN RELEASE 3.16.0	<code>http://<hostname>:<port>/service/rest/v1/status/writable</code>

SSL Termination

NXRM can be configured to serve content using SSL (see [Configuring to Serve Content via HTTPS³⁸](#)). The recommended approach for HA-C is to use the load balancer for SSL termination as it alleviates the need to configure certificates on each NXRM node.

³⁸ <https://help.sonatype.com/display/NXRM3/Configuring+SSL#ConfiguringSSL-InboundSSL-ConfiguringtoServeContentviaHTTPS>