# Hazelcast Deployment and Operations Guide

Version 4.2

# Hazelcast Deployment and Operations Guide for 4.2


1. Introduction . . . 1
   1.1. Purpose of This Document . . . 3
   1.2. Hazelcast Versions . . . 3
2. License Options and Management . . . 3
   2.1. Hazelcast OpenSource License . . . 3
   2.2. Hazelcast IMDG Enterprise License . . . 3
   2.3. Hazelcast IMDG Enterprise Licenses . . . 4
   2.4. Hazelcast IMDG Enterprise HD License . . . 4
   2.5. Setting Up IMDG Enterprise License in Your Environment . . . 4
   2.6. Accessing Licensing Information on your System . . . 5
   2.7. Expired License . . . 6
   2.8. How to Upgrade or Renew Your License . . . 6
3. Upgrading to Hazelcast 4.0 . . . 7
   3.1. Using the Migration Tool to Move Your Data to a 4.0 Cluster (Enterprise and Enterprise HD) . . . 7
4. Guidance for Specific Operating Environments . . . 8
   4.1. Solaris Sparc . . . 8
   4.2. VMWare ESX . . . 8
   4.3. Amazon Web Services . . . 9
   4.4. Windows . . . 9
5. Network Architecture and Configuration . . . 9
   5.1. Topologies . . . 9
   5.2. Advantages of Embedded Architecture . . . 12
   5.3. Advantages of Client-Server Architecture . . . 12
   5.4. Network Configuration . . . 12
   5.5. Open Binary Client Protocol . . . 15
   5.6. Partition Grouping . . . 15
   5.7. Cluster Discovery Protocols . . . 15
   5.8. WAN Replication (Enterprise Feature) . . . 17
   5.9. Advanced Network Configuration) . . . 18
6. Lifecycle, Maintenance and Updates . . . 19
   6.1. Configuration Management . . . 19
   6.2. Cluster Startup . . . 20
   6.3. Cluster Failover (Enterprise Feature) . . . 20
   6.4. Hot Restart Store (Enterprise HD Feature) . . . 20
   6.5. Cluster Scaling: Joining and Leaving Nodes . . . 27
   6.6. Health Check of Hazelcast IMDG Nodes . . . 28
   6.7. Shutting Down Hazelcast IMDG Nodes . . . 29

6.8. Maintenance and Software Updates . . . 30
6.9. Hazelcast IMDG Software Updates . . . 32
7. Understanding Configuration Overview . . . 33
7.1. Planning for a Stable Configuration Architecture . . . 35
7.2. Dynamic Configuration of Hazelcast 4.x . . . 36
8. Starting and Stopping Cluster Runbook . . . 36
8.1. Starting and Stopping Hazelcast . . . 36
8.2. Starting Hazelcast . . . 36
8.3. Stopping Hazelcast . . . 37
8.4. Hazelcast Cluster Script . . . 37
8.5. Hazelcast CP Subsystem Startup and Stop . . . 37
9. Cluster Sizing . . . 37
9.1. Sizing Considerations . . . 38
9.2. Example: Sizing a Cache Use Case . . . 39
10. Security & Hardening . . . 40
10.1. Features (Enterprise and Enterprise HD) . . . 40
10.2. Validating Secrets Using Strength Policy . . . 42
10.3. Security Defaults . . . 44
10.4. Advanced networking . . . 45
10.5. Hardening Recommendations . . . 46
10.6. Secure Context . . . 48
11. Deployment and Scaling Runbook . . . 49
11.1. Installation . . . 49
11.2. Pre-Requisites . . . 49
11.3. Java JDK . . . 49
11.4. License on the System . . . 49
11.5. Hazelcast Configuration file . . . 50
12. Starting and Stopping Hazelcast . . . 50
13. Cluster Discovery Basics . . . 51
14. Smart Clients . . . 51
15. Hazelcast Cloud Deployment Discovery Runbook . . . 53
16. Cloud Deployment . . . 53
17. Deployment and Scaling Runbook . . . 53
18. Installation . . . 53
19. Pre-Requisites . . . 53
20. Java JDK . . . 54
21. License on the System . . . 54
22. Hazelcast Configuration file . . . 54
23. Starting and Stopping Hazelcast . . . 55
24. Cluster Discovery Basics . . . 55
25. Smart Clients . . . 56

26. Cluster Audit and Monitoring . . . 56
27. CP Subsystem Overview . . . 57
27.1. CP Subsystem Example Declarative Configuration: . . . 58
27.2. CP Subsystem Persistence . . . 59
28. Rolling Member Upgrades . . . 59
28.1. Rolling Upgrades Compatibility . . . 59
28.2. Rolling Upgrade Procedure . . . 59
28.3. Upgrading Cluster Version . . . 60
28.4. Enabling Auto-Upgrading . . . 60
28.5. Network Partitions and Rolling Upgrades . . . 61
29. Failure Detection and Recovery . . . 61
29.1. Common Causes Of Node Failure . . . 62
29.2. Failure Detection on Cluster Members . . . 62
29.3. Health Monitoring & Alerts . . . 63
29.4. Recovery from a Partial or Total Failure . . . 65
29.5. Recovery from Client Connection Failures . . . 66
30. Logging Overview . . . 66
30.1. Managing your Logs . . . 67
30.2. Hazelcast IMDG Diagnostics Log . . . 68
30.3. Enabling Garbage Collection Logs . . . 72
31. Performance Tuning & Optimization . . . 75
31.1. Dedicated, Homogeneous Hardware Resources . . . 75
31.2. Partition Count . . . 75
31.3. Parallel Migration . . . 76
31.4. Dedicated Network Interface Controller for Hazelcast IMDG Members . . . 76
31.5. Network Settings . . . 76
31.6. Garbage Collection . . . 77
31.7. High-Density Memory Store (Enterprise HD Feature) . . . 78
31.8. Azul Zing® and Zulu® Support (Enterprise Feature) . . . 79
31.9. Pipelining . . . 79
31.10. Optimizing Queries . . . 79
31.11. Optimizing Serialization . . . 80
31.12. Serialization Optimization Recommendations . . . 82
31.13. Executor Service Optimizations . . . 83
31.14. Executor Service Tips and Best Practices . . . 84
31.15. Back Pressure . . . 85
31.16. Entry Processors . . . 85
31.17. Near Cache . . . 86
31.18. Client Executor Pool Size . . . 86
31.19. Clusters with Many (Hundreds) of Nodes or Clients . . . 87
31.20. Linux Memory Management Recommendations . . . 87

31.21. Basic Optimization Recommendations . . . 88
31.22. Setting Internal Response Queue Idle Strategies . . . 88
31.23. TLS/SSL Performance Improvements for Java . . . 89
31.24. AWS Deployments . . . 89
32. Management Center (Subscription and Enterprise Features) . . . 89
32.1. Cluster-Wide Statistics and Monitoring . . . 89
32.2. Web Interface Homepage . . . 89
32.3. Data Structure and Member Management . . . 93
32.4. Monitoring Cluster Health . . . 93
32.5. Management Center Configuration Tool . . . 94
32.6. Metrics Persistence . . . 95
32.7. Prometheus Exporter . . . 96
32.8. Monitoring WAN Replication . . . 97
32.9. Management Center Deployment . . . 98
33. Enterprise Cluster Monitoring with JMX & REST (Subscription and Enterprise Feature) . . . 100
33.1. Recommendations for Setting Alerts . . . 101
33.2. Actions and Remedies for Alerts . . . 101
34. Handling Network Partitions . . . 101
34.1. Split-Brain on Network Partition . . . 102
34.2. Split-Brain Protection . . . 104
34.3. Split-Brain Resolution . . . 106
35. How to Report Issues to Hazelcast . . . 108
35.1. Hazelcast Support Subscribers . . . 108
35.2. Hazelcast IMDG Open Source Users . . . 109

Welcome to the Hazelcast Deployment and Operations Guide. This guide includes concepts, instructions, and samples to guide you on how to properly deploy and operate on Hazelcast IMDG.

# 1. Introduction

This the Hazelcast IMDG (In-Memory Data Grid) Operations and Deployment Manual. This Manual is developed as a quick reference for general operations and deployment of Hazelcast Clusters. It is meant to be used in conjunction and in consistency with the More detailed Hazelcast Reference Manual. Hazelcast is an open source In-Memory Data Grid (IMDG) that provides fast scalable distributed In-Memory computing and application performance. Hazelcast makes distributed computing leverages the concurrency and distributed implementations of many interfaces from Java such as Map, Queue, ExecutorService, Lock and JCache. Hazelcast also provides a convenient set of APIs to access the CPUs in your cluster for maximum processing speed. Hazelcast is designed to be lightweight and easy to use and is delivered as a compact library (JAR) and has no external dependencies other than Java.

Distributed applications can leverage Hazelcast for distributed caching, synchronization, clustering, processing, pub/sub messaging, etc. Hazelcast is implemented in Java and has clients for Java, C/C++, .NET, REST, Python, Go and Node.js. Hazelcast also speaks Memcached protocol. It plugs into Hibernate and can easily be used with any existing database system.

If you are looking for in-memory speed, elastic scalability and the developer friendliness of NoSQL, Hazelcast is a great choice.

**Hazelcast is Simple**

Hazelcast is written in Java with no other dependencies. It exposes the same API from the familiar Java util package, exposing the same interfaces. Just add hazelcast.jar to your classpath and you can quickly enjoy JVMs clustering and start building scalable applications.

**Hazelcast is Peer-to-Peer**

Unlike many NoSQL solutions, Hazelcast is peer-to-peer. There is no master and slave; there is no single point of failure. All members store equal amounts of data and do equal amounts of processing. You can embed Hazelcast in your existing application or use it in client and server mode where your application is a client to Hazelcast members.

**Hazelcast is Scalable**

Hazelcast is designed to scale up to hundreds and thousands of members. Simply add new members; they automatically discover the cluster and linearly increase both the memory and processing capacity. The members maintain a TCP connection between each other and all communication is performed through this layer.

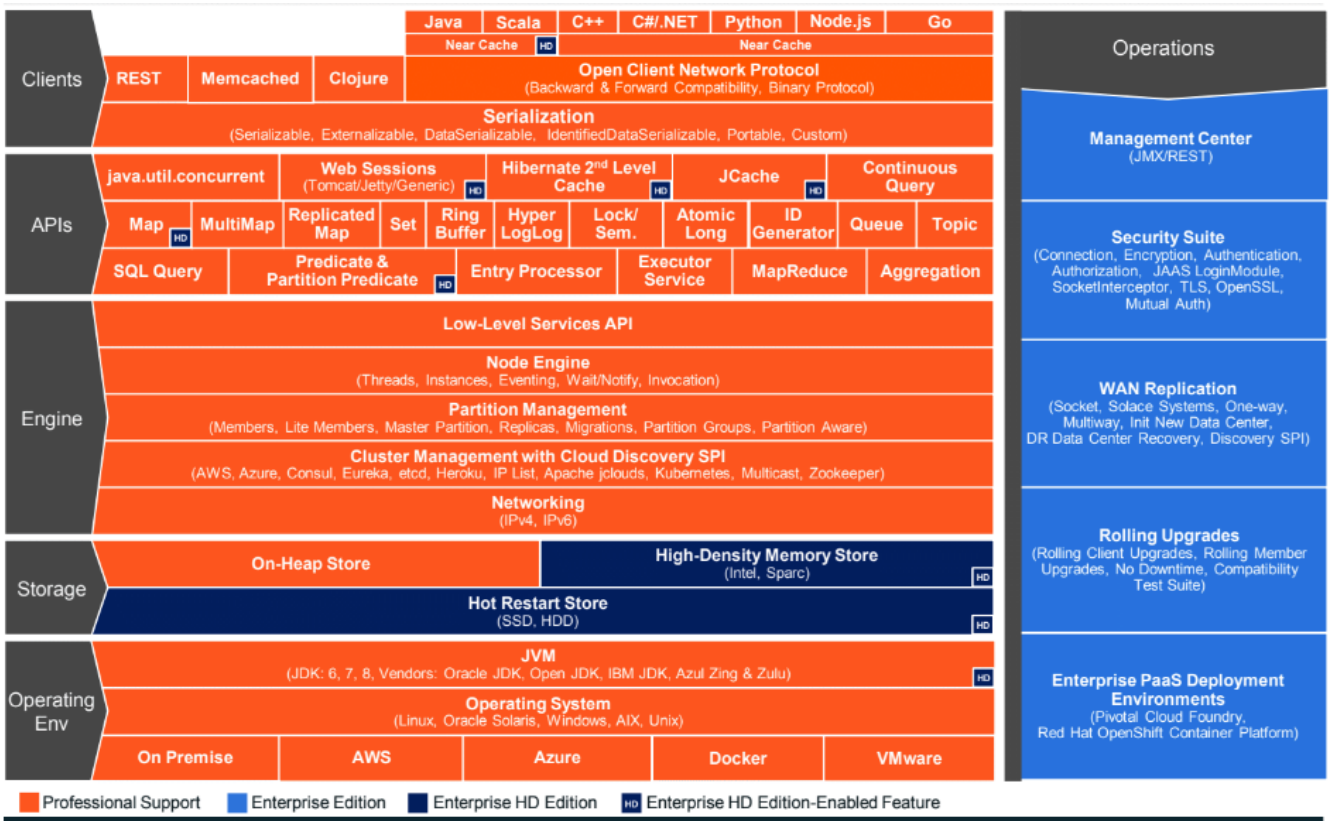
**Hazelcast is Fast**

Hazelcast stores everything in-memory. It is designed to perform very fast reads and updates.

### Hazelcast is Redundant

Hazelcast keeps the backup of each data entry on multiple members. On a member failure, the data is restored from the backup and the cluster continues to operate without downtime.

Hazelcast IMDG provides a convenient, familiar, and powerful interface for developers to work with distributed data structures and other aspects of in-memory computing. For example, in its simplest form Hazelcast can be treated as an implementation of a thread-safe key-value data structure that can be accessed from multiple nodes either on the same machine or distributed in the network, or both. However, the Hazelcast IMDG architecture has both the flexibility and the advanced features required to be useful in a large number of different architectural patterns and styles. The following schematic represents the basic architecture of Hazelcast IMDG.




Though Hazelcast IMDG's architecture is sophisticated, many users are happy to integrate at the level of the `java.util.concurrent` or `javax.cache` APIs.

The core Hazelcast IMDG technology:

- Is open source
- Is written in Java
- Supports Java 8 SE (See detailed info at Supported JVMs)
- Uses minimal dependencies
- Has simplicity as a key concept

The primary capabilities that Hazelcast IMDG provides include:

- Elasticity

- Redundancy
- High performance

Elasticity means that Hazelcast IMDG clusters can increase or reduce capacity simply by adding or removing nodes. Redundancy is controlled via a configurable data replication policy (which defaults to one synchronous backup copy). To support these capabilities, Hazelcast IMDG uses the concept of members. Members are JVMs that join a Hazelcast IMDG cluster. A cluster provides a single extended environment where data can be synchronized between and processed by its members.

## 1.1. Purpose of This Document

If you are a Hazelcast IMDG user planning to go into production with a Hazelcast IMDG-backed application, or you are curious about the practical aspects of deploying and running such an application, this guide will provide an introduction to the most important aspects of deploying and operating a successful Hazelcast IMDG installation.

In addition to this guide, there are many useful resources available online including Hazelcast IMDG product documentation, Hazelcast forums, books, webinars, and blog posts. Where applicable, each section of this document provides links to further reading if you would like to delve more deeply into a particular topic.

Hazelcast also offers support, training, and consulting to help you get the most out of the product and to ensure successful deployment and operations. Visit pricing for more information.

## 1.2. Hazelcast Versions

This document is current to Hazelcast IMDG version 4.2. It is not explicitly backward-compatible to earlier versions, but may still substantially apply. Hazelcast version 4.2 is a breaking version from previous version of Hazelcast IMDG e.g. 3.x.x. and therefore because of a significant deprecation of feature from 3.x.x to 4.x and a refactoring of other major features.

# 2. License Options and Management

Hazel cast offer 3 different Licensing options as detailed below.

## 2.1. Hazelcast OpenSource License

Hazelcast OpenSource is distributed using the Apache License 2, therefore permissions are granted to use, reproduce and distribute it along with any kind of open source and closed source applications.

## 2.2. Hazelcast IMDG Enterprise License

Hazelcast IMDG Enterprise is a commercial product of Hazelcast, Inc. and is distributed under a commercial license that must be acquired before using it in any type of released software. If you

are running Hazelcast IMDG Enterprise you must make sure you have the correct license acquired and purchased. Licensing depends on version of IMDG, type of license and number of nodes. These must be correctly defined in order for you to run with a valid and proper license Enterprise license key. Review your features uses carefully because if you need to use some advanced features you will need to validate that you have the corresponding valid license key.

If you plan on using the following features you will need Hazelcast Enterprise.

## 2.3. Hazelcast IMDG Enterprise Licenses

Hazelcast IMDG Enterprise offers you two types of licenses: Enterprise and Enterprise HD. The supported features differ in your Hazelcast setup according to the license type you own.

Enterprise features are the following:

- Security
- WAN Replication
- Clustered REST
- Clustered JMX
- Striim Hot Cache
- Rolling Upgrades

## 2.4. Hazelcast IMDG Enterprise HD License

If you plan on using the following features you will need Hazelcast Enterprise HD License.

- High-Density Memory Store
- Hot Restart Persistence

## 2.5. Setting Up IMDG Enterprise License in Your Environment

Setting the License Key To use Hazelcast IMDG Enterprise, you need to set the provided license key using one of the configuration methods shown below.

Depending on the used feature-set, Hazelcast has certain runtime dependencies which might have different licenses. Following are dependencies and their respective licenses. Please contact Hazelcast sales department for more information on commercial offers.

If you have a valid license for Hazelcast IMDG Enterprise, you will receive a unique license key from Hazelcast Support that will enable the fullset of Hazelcast IMDG Enterprise capabilities. Ensure a valid license key file is available on the filesystem of each member node and configure the path to it using either declarative, programmatic, or Spring configuration. A fourth option is to set the following system property:

```
-Dhazelcast.enterprise.license.key=/path/to/license/key
```

You can also add add the license to hazelcast.xml configuration file as shown. **Declarative Configuration:**

```
<hazelcast>
    ...
    <license-key>Your Enterprise License Key</license-key>
    ...
</hazelcast>
```

**Programmatic Configuration:** Alternatively, you can set your license key programmatically as shown below.

```
Config config = new Config();
config.setLicenseKey( "Your Enterprise License Key" );
```

For more detailed Licensing information please review the following License Reference. **Further Reading:**

- Online Documentation, License Information: https://docs.hazelcast.org/docs/latest/manual/html-single/#licensing

## 2.6. Accessing Licensing Information on your System

Hazelcast offer a rest API that can leveraged for a number of functional uses and can be used to determine you Licensing details. You can obtain license information through JMX and REST API. The following data is available:

- `Max Node Count`: Maximum nodes allowed to form a cluster under the current license
- `Expiry Date`: The expiry date of the current license
- `Type Code`: The type code of the current license
- `Type`: The type of the current license
- `Owner Mail`: The email of the owner on the current license
- `Company Name`: The name of the company on the current license

You can review the API reference here: **Further Reading:**

- Online Documentation, License Information: https://docs.hazelcast.org/docs/latest/manual/html-single/#license-info

## 2.7. Expired License

**PLEASE NOTE IF YOU HAVE AN EXPIRED LICENSE YOUR SYSTEM WILL NOT START:** In order to maintain a functional environment it's strongly encouraged that you stay up to date on your license expiry period. You should keep of your license dates and features and check them often and request new licenses as needed before license expiry. You should also validate that you have the right license for the feature and number of nodes you intend to run.

If you have an expired License or if you current license is nearing expiry, Hazelcast will issue warnings about approaching license expiry in the logs with the following format:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ WARNING @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
HAZELCAST LICENSE WILL EXPIRE IN 29 DAYS.
Your Hazelcast cluster will stop working after this time.

Your license holder is customer@example-company.com, you should have them contact
our license renewal department, urgently on info@hazelcast.com
or call us on +1 (650) 521-5453

Please quote license id CUSTOM_TEST_KEY
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

You should carefully note and monitor your license warnings messages so that you can renew or acquire a valid license in a timely manner in order to avoid interruptions to your Hazelcast Instances. **Further Reading:**

- Online Documentation, License Information: https://docs.hazelcast.org/docs/latest/manual/html-single/index.html#license-info

## 2.8. How to Upgrade or Renew Your License

If you wish to upgrade your license or renew your existing license before it expires, contact Hazelcast Support to receive a new license. To install the new license, replace the license key on each member host and restart each node, one node at a time, similar to the process described in the "Live Updates to Cluster Member Nodes" section above.



If your license expires in a running cluster or Management Center, do not restart any of the cluster members or the Management Center JVM. Hazelcast IMDG will not start with an expired or invalid license. Reach out to Hazelcast Support to resolve any issues with an expired license.

**Further Reading:**

- Online Documentation, Installing Hazelcast IMDG Enterprise

# 3. Upgrading to Hazelcast 4.0

IMDG 4.0 is a major version release. The last major version release was over five years ago. Major releases contain breaks in compatibility various areas such as the wire protocols and API for example, as well as removing the previously deprecated API. Due to the breaking changes that have been made to the client and cluster member protocols, it is not possible to perform any in-place or rolling upgrade from a running IMDG 3.x cluster to IMDG 4.x. The only way to upgrade to IMDG 4.x is to completely shutdown your existing 3.x.x cluster and stand up a new 4.x cluster. If you have a license for WAN Replication, you do have the ability to migrate your `IMap` and `ICache` data using the migration tool. This process is explained below.

**Further Reading:** Upgrading to Hazelcast IMDG 4.0

## 3.1. Using the Migration Tool to Move Your Data to a 4.0 Cluster (Enterprise and Enterprise HD)

The Migration Tool uses Hazelcast's own WAN Replication feature to replicate data from a 3.12.x source cluster to a special "migration" member in a 4.x target. This member is running Hazelcast with a migration jar that contains the compatibility code needed to accept connection from the 3.12.x cluster. The migration member can then distribute the data to the proper partitions on the normal members (similar to a typical WAN replication).

**The following is an example of a common migration scenario:**

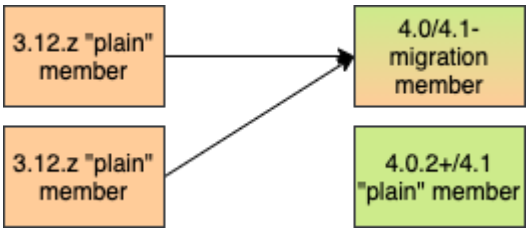



*Figure 1. Migrating 3.12 → 4.0/4.1 (ACTIVE/PASSIVE)*

1. **Configure the 3.12.x Cluster:** The source cluster needs to be configured for WAN Replication and a WAN Publisher needs to be setup with the 4.x migration member(s) as target endpoints. See WAN Replication for more information.
2. **Configure the 4.x Cluster:** The 4.x target cluster needs to have *at least one* migration member. These can be regular or lite members. Using a lite member will simplify the latter steps.
3. **Migrate Data:** Once both clusters have been started and WAN Replication has been configured, a manual WAN Sync can be performed (`IMap` only). Alternatively, you can wait for enough entries to be replicated (`IMap` or `ICache`).
4. **Shutdown 3.12.x Cluster:** Once the data has be successfully replicated to the 4.x cluster, the 3.12.x cluster can be shutdown.
5. **Remove Migration Members:** Finally, you can shutdown the migration member(s) and optionally replace them with a "non-migration" 4.x member. Here is the advantage of using lite migration members, since shutting them down does not trigger trigger partition migrations.

### 3.1.1. Other Considerations

- Migrations can also be performed from 4.x to 3.12.x. In this case the 3.12.x cluster needs the member(s) with the migration compatibility code. This also means that Active-Active migration scenarios are possible.
- Data migration is limited to `IMap` and `ICache` data structures
- The "migration" member needs to be able to deserialize and serialize all received keys and values
- The "non-migration" members in the 4.x cluster need to be at least 4.0.2 or 4.1 (any patch version)
- The members of the 3.12 cluster can have any patch version but need to at least be 3.12
- This feature is only available with an Enterprise license
- These special migration jars can be provided upon request by Hazelcast Support
- Additional details and instructions regarding the migration tool can be found in the documentation included with the migration artifact under Appendix B.1: *Migrating Between IMDG 3.12 and 4.0/4.1 Releases Using the Migration tool*

# 4. Guidance for Specific Operating Environments

Hazelcast IMDG works in many operating environments. Some environments have unique considerations. These are highlighted below.

## 4.1. Solaris Sparc

Hazelcast IMDG Enterprise HD is certified for Solaris Sparc starting with Hazelcast IMDG Enterprise HD version 3.6. Versions prior to that have a known issue with HD Memory due to the Sparc architecture not supporting unaligned memory access.

## 4.2. VMWare ESX

Hazelcast IMDG is certified on VMWare VSphere 5.5/ESXi 6.0.

Generally speaking, Hazelcast IMDG can use all of the resources on a full machine. Splitting a single physical machine into multiple VMs and thereby dividing resources is not required.

### 4.2.1. Best Practices

- Avoid over-committing memory. Always use dedicated physical memory for guests running Hazelcast IMDG.
- Do not use Memory Ballooning.
- Be careful over-committing CPU. Watch for CPU Steal Time.

- Do not move guests while Hazelcast IMDG is running. Disable vMotion (* see next section).
- Always enable verbose GC logs - when "Real" time is higher then "User" time then it may indicate virtualization issues. JVM is off CPU during GC (and probably waiting for I/O).
- Note VMWare guests network types.
- Use Pass-through harddisks / partitions. Do not to use image-files.
- Configure Partition Groups to use a separate underlying physical machine for partition backups.

### 4.2.2. Common VMWare Operations with Known Issues

- Live migration (vMotion). First stop Hazelcast IMDG. Restart after the migration.
- Automatic Snapshots. First stop Hazelcast IMDG and restart after the snapshot.

### 4.2.3. Known networking issues

- Network performance issues, including timeouts, might occur with LRO (Large Receive Offload) enabled on Linux virtual machines and ESXi/ESX hosts. We have specifically had this reported in VMware environments, but it could potentially impact other environments as well. We strongly recommend disabling LRO when running in virtualized environments: https://kb.vmware.com/s/article/1027511

## 4.3. Amazon Web Services

See our dedicated AWS Deployment Guide: Amazon Deployment Guide

## 4.4. Windows

According to a reported rare case, IO threads can consume a lot of CPU cycles unexpectedly, even in an idle state. This can lead to CPU usage going up to 100%. This is reported not only for Hazelcast but for other github projects as well. Workaround for such cases is to supply system property `-Dhazelcast.io.selectorMode=selectwithfix` on JVM startup. Please see related github issue for more details: https://github.com/hazelcast/hazelcast/issues/7943#issuecomment-218586767

# 5. Network Architecture and Configuration

## 5.1. Topologies

You can deploy a Hazelcast IMDG cluster in two modes of operation: Embedded or Client/Server.

If you have an application whose main focal point is asynchronous or high performance computing and lots of task executions, then Embedded deployment is the preferred way. In an embedded deployment, each member (JVM) includes both the application and Hazelcast IMDG services and data. The advantage of the Embedded deployment is having a low-latency data access.

In a client-server deployment, Hazelcast IMDG services and data are centralized on one or more

members and are accessed by the application through clients. You can have a cluster of server members that can be independently created and scaled. Your clients communicate with these members to reach to Hazelcast data and services on them.

These two topology approaches are illustrated in the following diagrams.

Here is the embedded approach:

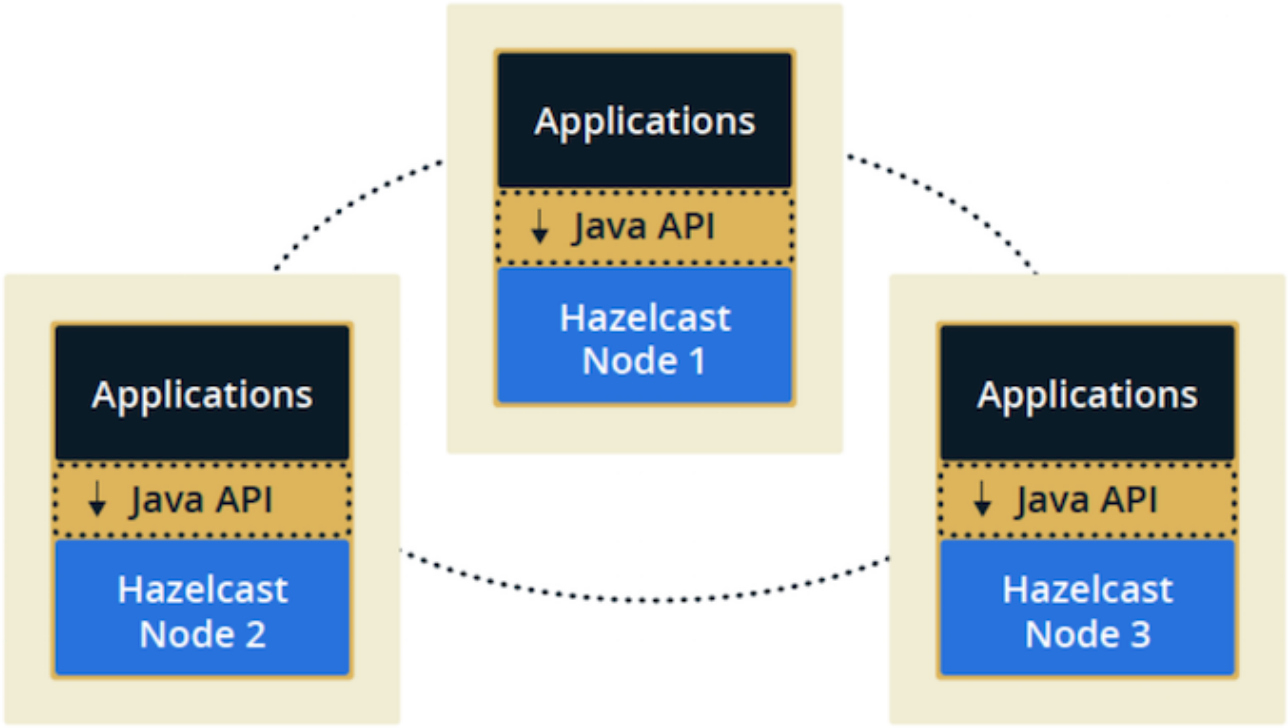



*Figure 2. Hazelcast IMDG Embedded Topology*

And the client-server topology:

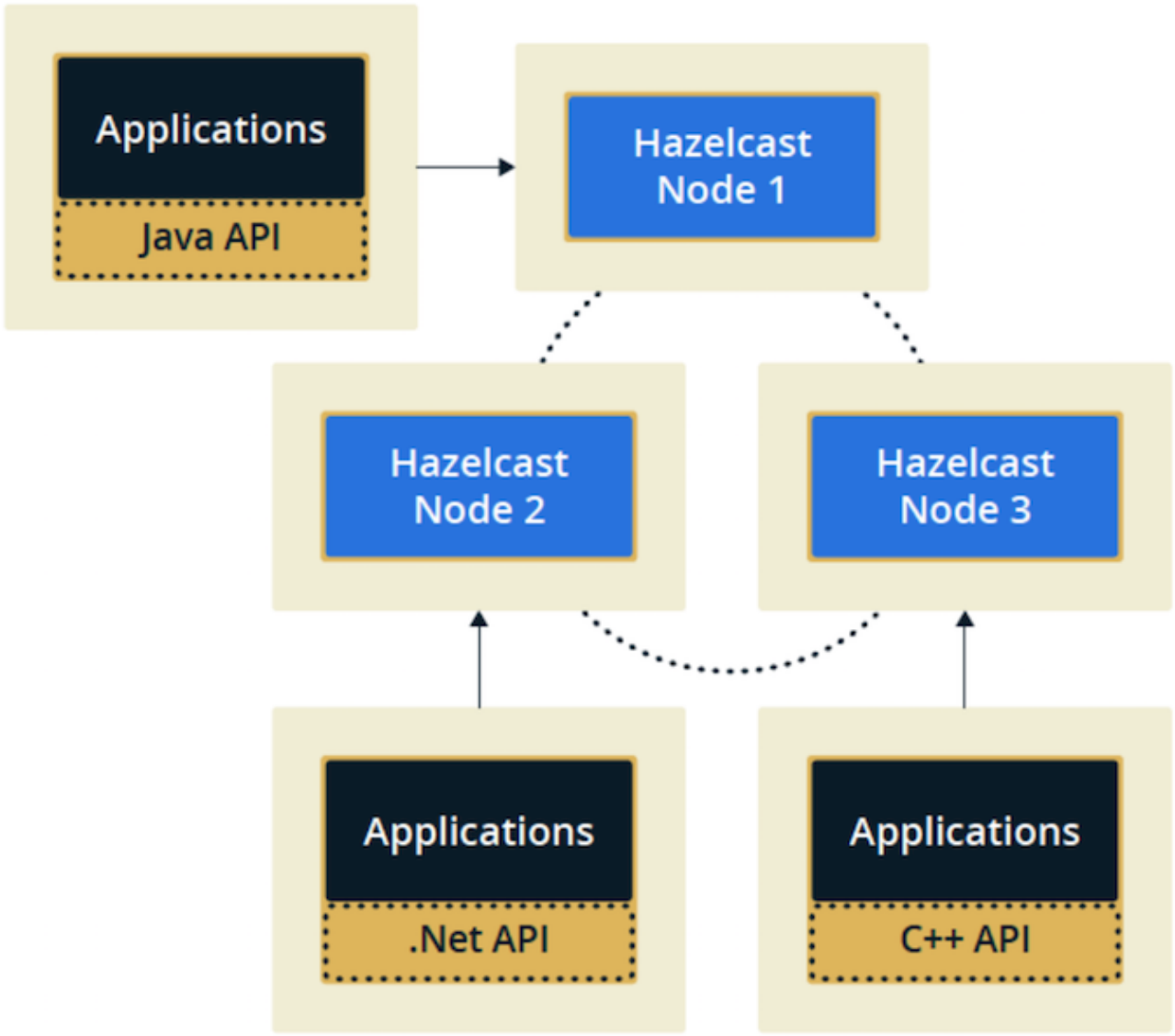



*Figure 3. Hazelcast IMDG Client-Server Topology*

Under most circumstances, we recommend the client-server topology, as it provides greater flexibility in terms of cluster mechanics. For example, member JVMs can be taken down and restarted without any impact on the overall application. The Hazelcast IMDG client will simply reconnect to another member of the cluster. Client-server topologies isolate application code from purely cluster-level events.

Hazelcast IMDG allows clients to be configured programmatically within the client code, by XML or YAML, or by properties files. Clients have quite a few configurable parameters, including known members of the cluster. Hazelcast IMDG will discover the other members as soon as they are online, but they need to connect first. In turn, this requires the user to configure enough addresses to ensure that the client can connect to the cluster somewhere.

In production applications, the Hazelcast IMDG client should be reused between threads and operations. It is designed for multithreaded operation. Creation of a new Hazelcast IMDG client is relatively expensive, as it handles cluster events, heartbeating, etc., so as to be transparent to the user.

## 5.2. Advantages of Embedded Architecture

The main advantage of using the embedded architecture is its simplicity. Because the Hazelcast IMDG services run in the same JVMs as the application, there are no extra servers to deploy, manage, or maintain. This simplicity especially applies when the Hazelcast IMDG cluster is directly tied to the embedded application.

## 5.3. Advantages of Client-Server Architecture

For most use cases, however, there are significant advantages to using the client-server architecture. Broadly, they are as follows:

- Cluster member lifecycle is independent of application lifecycle
- Resource isolation
- Problem isolation
- Shared infrastructure
- Better scalability

## 5.4. Network Configuration

Hazelcast cluster depend on a stable well configured and resourced network. Network configuration and related configurations are set via the network element in the Hazelcast XML configuration file or the class NetworkConfig when using programmatic configuration. Its is an essential part of managing your Hazelcast Cluster to document and understand your network configuration settings and details. In addition deploying on cloud environments by such providers e.g. Amazon, Azure, Google, etc require that you understand how to properly manage configure and monitor your network in cloud environments.

Here are some of the elements to keep track of about your cluster. These elements are crucial in monitoring and provide details about your cluster topology in issue scenarios

- Public Address : public-address overrides the public address of a member. By default, a member selects its socket address as its public address. But behind a network address translation (NAT), two endpoints (members) may not be able to see/access each other. If both members set their public addresses to their defined addresses on NAT, then that way they can communicate with each other. In this case, their public addresses are not an address of a local network interface but a virtual address defined by NAT.
- Port Number : You can set the ports that Hazelcast uses to communicate between cluster members. The default value is 5701. Make sure that the desired ports are open and available.
- Outbound Ports : By default, Hazelcast picks up an ephemeral port during socket bind operation. You should review and understand your security policies, firewalls and other network options that may require you to restrict outbound ports that are used by Hazelcast-enabled applications. You can limit and configure Hazelcast to use only defined outbound ports in your configuration file.
- Reuse Address : When you shutdown a cluster member, the server socket port goes into the

TIME_WAIT state for the next couple of minutes. If you start the member right after shutting it down, you may not be able to bind it to the same port because it is in the TIME_WAIT state. If you set the reuse-address element to true, the TIME_WAIT state is ignored and you can bind the member to the same port again. It is recommended to have a custom script that can be run post shutdown to validate that network resources have been relinquished at shutdown and also to validate network usage post startup and audit socket availability and acceptance.

- Join : As mentioned earlier Hazelcast uses auto discovery mechanism to find and join a cluster. This is determined by the configured join configuration element. Hazelcast provides multicast, TCP/IP, EC2 and jclouds® discovery mechanisms.
- Interfaces : You can specify which network interfaces that Hazelcast should use. Servers mostly have more than one network interface, so you its recommended to list the valid IPs. If network interface configuration is enabled (it is disabled by default) and if Hazelcast cannot find a matching interface, then it prints a message on the console and does not start on that member.
- IPv6 Support : Hazelcast supports IPv6 addresses seamlessly (This support is switched off by default; If needed please review the Reference documentation on how to enable and configure support for IPv6).
- Member Address Provider SPI : By default, Hazelcast chooses the public and bind address. You can influence on the choice by defining a public-address in the configuration or by using other properties mentioned above. In some cases, though, these properties are not enough and the default address picking strategy chooses wrong addresses. This may be the case when deploying Hazelcast in some cloud environments, such as AWS, when using Docker or when the instance is deployed behind a NAT and the public-address property is not enough (see the Public Address section). The Member Address Provider SPI is not intended to provide addresses of other cluster members with which the Hazelcast instance forms a cluster.

**Cluster Member Node Lifecycle Independent of Application Lifecycle**

The practical lifecycle of Hazelcast IMDG member nodes is usually different from any particular application instance. When Hazelcast IMDG is embedded in an application instance, the embedded Hazelcast IMDG node will be started and shut down alongside its co-resident application instance, and vice-versa. This is often not ideal and may lead to increased operational complexity. When Hazelcast IMDG nodes are deployed as separate server instances, they and their client application instances may be started and shut down independently.

**Resource Isolation**

When Hazelcast IMDG is deployed as a member on its own dedicated host, it does not compete with the application for CPU, memory, and I/O resources. This makes Hazelcast IMDG performance more predictable and reliable.

**Easier Problem Isolation**

When Hazelcast IMDG member activity is isolated to its own server, it’s easier to identify the cause of any pathological behavior. For example, if there is a memory leak in the application causing unbounded heap usage growth, the memory activity of the application is not obscured by the co-resident memory activity of Hazelcast IMDG services. The same holds true for CPU and I/O issues. When application activity is isolated from Hazelcast IMDG services, symptoms are automatically isolated and easier to recognize.

**Shared Infrastructure**

The client-server architecture is appropriate when using Hazelcast IMDG as a shared infrastructure used by multiple applications, especially those under the control of different workgroups.

**Better Scalability**

The client-server architecture has a more flexible scaling profile. When you need to scale, simply add more Hazelcast IMDG servers. With the client-server deployment model, client and server scalability concerns may be addressed independently.

**Lazy Initiation and Connection Strategies**

Starting with version 3.9, you can configure the Hazelcast IMDG client's starting mode as async or sync using the configuration element `async-start`. When it is set to true (async), Hazelcast IMDG will create the client without waiting for a connection to the cluster. In this case, the client instance will throw an exception until it connects to the cluster. If `async-start` is set to false, the client will not be created until the cluster is ready to use clients and a connection with the cluster is established. The default value for `async-start` is false (sync).

Again starting with Hazelcast IMDG 3.9, you can configure how the Hazelcast IMDG client will reconnect to the cluster after a disconnection. This is configured using the configuration element reconnect-mode. It has three options: OFF, ON or ASYNC.

- The option OFF disables the reconnection.
- ON enables reconnection in a blocking manner where all the waiting invocations will be blocked until a cluster connection is established or failed. This is the default value.
- The option ASYNC enables reconnection in a non-blocking manner where all the waiting invocations will receive a `HazelcastClientOfflineException`.

Starting from version 3.11, you can also fine tune client's connection retry behavior. You can apply an exponential backoff instead of a periodic retry with a fixed count of attempt limit. This is done through `connection-retry` element when configuring declaratively or through the object `ConnectionRetryConfig` when configuring programmatically.

**Further Reading:**

- Online Documentation: Java Client Connection Strategy
- Online Documentation: Configuring Client Connection Retry

**Achieve Very Low Latency with Client-Server** If you need very low latency data access, but you also want the scalability advantages of the client-server deployment model, consider configuring the clients to use Near Cache. This will ensure that frequently used data is kept in local memory on the application JVM.

**Further Reading:**

- Online Documentation, Near Cache: Near Cache.

## 5.5. Open Binary Client Protocol

Hazelcast IMDG includes an Open Binary Protocol to facilitate the development of Hazelcast IMDG client APIs on any platform. In addition to the protocol documentation itself, there is an implementation guide and a Python client API reference implementation that describes how to implement a new Hazelcast IMDG client.

**Further Reading:**

- Online Documentation, Open Binary Client Protocol
- Online Documentation, Client Protocol Implementation Guide

## 5.6. Partition Grouping

By default, Hazelcast IMDG distributes partition replicas randomly and equally among the cluster members, assuming all members in the cluster are identical. But for cases where all members are not identical and partition distribution needs to be done in a specialized way, Hazelcast provides the following types of partition grouping:

- **HOST_AWARE:** You can group members automatically using the IP addresses of members, so members sharing the same network interface will be grouped together. This helps to avoid data loss when a physical server crashes because multiple replicas of the same partition are not stored on the same host.
- **CUSTOM:** Custom grouping allows you to add multiple differing interfaces to a group using Hazelcast IMDG's interface matching configuration.
- **PER_MEMBER:** You can give every member its own group. This provides the least amount of protection and is the default configuration.
- **ZONE_AWARE:** With this partition group type, Hazelcast IMDG creates the partition groups with respect to member attributes map entries that include zone information. That means backups are created in the other zones and each zone will be accepted as one partition group. You can use ZONE_AWARE configuration with Hazelcast AWS, Hazelcast Azure, Hazelcast GCP, or Hazelcast Kubernetes Discovery Service plugins.

When using the ZONE_AWARE partition grouping, a Hazelcast cluster spanning multiple AZs should have an equal number of members in each AZ. Otherwise, it will result in uneven partition distribution among the members.

- **Service Provide Interface (SPI):** You can provide your own partition group implementation using the SPI configuration.

**Further Reading:**

Online Documentation, Partition Group Configuration

## 5.7. Cluster Discovery Protocols

Hazelcast IMDG supports the following options for cluster creation and discovery when nodes start:

- Multicast
- TCP
- Pluggable Cloud Discovery Service Provider Interface (i.e. AWS, Azure, GCP, or Kubernetes/OpenShift)

Once a node has joined a cluster, all further network communication is performed via TCP.

### 5.7.1. Multicast

The advantage of multicast discovery is its simplicity and flexibility. As long as Hazelcast IMDG's local network supports multicast, the cluster members do not need to know each other's specific IP addresses when they start. This is especially useful during development and testing. In production environments, if you want to avoid accidentally joining the wrong cluster, then use Group Configuration.

We do not generally recommend multicast for production use. This is because UDP is often blocked in production environments and other discovery mechanisms are more definite.

**Further Reading:**

Online Documentation, Group Configuration

### 5.7.2. TCP

When using TCP for cluster discovery, the specific IP address of at least one other cluster member must be specified in the configuration. Once a new node discovers another cluster member, the cluster will inform the new node of the full cluster topology, so the complete set of cluster members need not be specified in the configuration. However, we recommend that you specify the addresses of at least two other members in case one of those members is not available at start.

### 5.7.3. Cloud Discovery SPI

Hazelcast IMDG provides a Cloud Discovery Service Provider Interface (SPI) to allow for pluggable, third-party discovery implementations. The plugins allow Hazelcast cluster members to discover each other on the cloud platforms.

For more information please check Online Documentation, Cloud Discovery Plugins.

**Further Reading:**

For detailed information on cluster discovery and network configuration for Multicast, TCP, and Discovery SPI see the following documentation:

- Mastering Hazelcast IMDG, Network Configuration
- Online Documentation, Hazelcast IMDG Cluster Discovery
- Online Documentation, Hazelcast IMDG Discovery SPI

### 5.7.4. Firewalls, NAT, Network Interfaces and Ports

Hazelcast IMDG's default network configuration is designed to make cluster startup and discovery simple and flexible out of the box. It's also possible to tailor the network configuration to fit the specific requirements of your production network environment.

If your server hosts have multiple network interfaces, you may customize the specific network interfaces Hazelcast IMDG should use. You may also restrict which hosts are allowed to join a Hazelcast IMDG cluster by specifying a set of trusted IP addresses or ranges. If your firewall restricts outbound ports, you may configure Hazelcast IMDG to use specific outbound ports allowed by the firewall. Nodes behind network address translation (NAT) in, for example, a private cloud may be configured to use a public address.

**Further Reading:**

- Mastering Hazelcast IMDG eBook, Network Configuration
- Online Documentation, Network Configuration
- Online Documentation, Network Interfaces
- Online Documentation, Outbound Ports

## 5.8. WAN Replication (Enterprise Feature)

If, for example, you have multiple data centers to provide geographic data locality or disaster recovery and you need to synchronize data across the clusters, Hazelcast IMDG Enterprise supports wide-area network (WAN) replication. WAN replication operates in either active-passive mode, where an active cluster backs up to a passive cluster, or active-active mode, where each participating cluster replicates to all others.

You may configure Hazelcast IMDG to replicate all data or restrict replication to specific shared data structures. In certain cases, you may need to adjust the replication queue size. The default replication queue size is 100,000, but in high volume cases, a larger queue size may be required to accommodate all of the replication messages.

When it comes to defining WAN Replication endpoints, Hazelcast IMDG offers two options:

- Using Static Endpoints: A straightforward option when you have fixed endpoint addresses.
- Using the Discovery SPI: Suitable when you want to use WAN Replication with endpoints on various cloud infrastructures (such as Amazon EC2) where the IP address is not known in advance. Several cloud plugins are already implemented and available. For more specific cases, you can provide your own discovery SPI implementation.



Discovery SPI for Amazon EC2 uses `DescribeInstances` API by AWS which might be limited on daily usage. You can decrease the amount of `DescribeInstances` calls by increasing the WAN Replication property `discovery.period` to a higher value in seconds.

Starting with Hazelcast IMDG 3.12, we have redesigned WAN Replication to allow tuning for lower

latencies and higher throughput. To that end, we have introduced several new WAN Replication parameters. While WAN Replication is sufficient with out-of-the-box settings in most cases, these new parameters can be used to improve WAN Replication performance depending on the use case. An in-depth explanation of these new parameters can be found in the *Tuning WAN Replication For Lower Latencies and Higher Throughput* section of the Hazelcast Reference Manual.

**Further Reading:**

- Online Documentation, WAN Replication
- Tuning WAN Replication For Lower Latencies and Higher Throughput

## 5.9. Advanced Network Configuration)

In previous older version of Hazelcast IMDG Hazelcast members use a single server socket for all kinds of connections. Cluster members and Hazelcast clients all implemented the Open Binary Client Protocol and HTTP protocol clients connect to a single server socket that handles all the protocols.

That changed starting with Hazelcast 3.12 and 4.0. With these newer versions you can configure the Hazelcast members with separate server sockets using a different network configuration for different protocols. This configuration scheme allows more flexibility when deploying Hazelcast as described in a variety of ways. e.g. for more secure network configuration.

For this additional benefit the cost is more complexity which requires a careful documentation of your network architecture in order to be able to debug your network traffic when necessary.

Please note that Advanced network configuration and single-socket network configuration are mutually exclusive. You can enable only one of this network configuration options.

When using advanced network configuration it is important to avoid the following configuration setups as they can lead to error an ambiguous behavior on the network. Please validate you network configuration often for the following scenarios.

- You cannot use multiplex protocols on a single advanced network endpoint? e.g, using a single server socket to listen for MEMBER and CLIENT protocols. each endpoint configuration that defines a server socket must bind to a different socket address.
- You cannot mix unisocket and advanced network members in the same cluster. This can lead to undefined network behavior.
- You cannot configure multiple server socket endpoints for the same protocol You can only configure multiple server socket endpoints for WAN protocol. For other protocols (MEMBER, CLIENT, REST, MEMCACHE), a single server socket can be configured.

If any of these configuration options are found in your network configuration please resolve them promptly.

**Further Reading:**

- Online Documentation, Advanced Network Configuration *

# 6. Lifecycle, Maintenance and Updates

When operating a Hazelcast IMDG installation over time, planning for certain lifecycle events will ensure high uptime and smooth operation. Before moving your Hazelcast IMDG application into production, you will want to have policies in place for handling various aspects of your installation such as:

- Changes in cluster and network configuration
- Startup and shutdown procedures
- Application, software, and hardware updates

## 6.1. Configuration Management

You can configure Hazelcast IMDG using one or more of the following options:

- Declaratively
- Programmatically
- Using Hazelcast IMDG system properties
- Within the Spring context
- Dynamically adding configuration on a running cluster (starting with Hazelcast IMDG 3.9)

Some IMap configuration options may be updated after a cluster has been started. For example, TTL and backup counts can be changed via the Management Center. Also, starting with Hazelcast IMDG 3.9, it is possible to dynamically add configuration for certain data structures at runtime. These can be added by invoking one of the corresponding `Config.addConfig` methods on the Config object obtained from a running member.

Other configuration options can't be changed on a running cluster. Hazelcast IMDG will not accept nor communicate any new configuration of joining nodes that differs from the existing cluster configuration. The following configurations will remain the same on all nodes in a cluster and may not be changed after cluster startup:

- Group name and password
- Application validation token
- Partition count
- Partition group
- Joiner

The use of a file change monitoring tool is recommended to ensure proper and identical configuration across the members of the cluster.

**Further Reading:**

- Online Documentation

- Mastering Hazelcast IMDG eBook

## 6.2. Cluster Startup

Hazelcast IMDG cluster startup is typically as simple as starting all of the nodes. Cluster formation and operation will happen automatically. However, in certain use cases you may need to coordinate the startup of the cluster in a particular way. In a cache use case, for example, where shared data is loaded from an external source such as a database or web service, you may want to ensure the data is substantially loaded into the Hazelcast IMDG cluster before initiating normal operation of your application.

**Data and Cache Warming**

A custom MapLoader implementation may be configured to load data from an external source either lazily or eagerly. The Hazelcast IMDG instance will immediately return lazy-loaded maps from calls to getMap(). Alternately, the Hazelcast IMDG instance will block calls to getMap() until all of the data is loaded from the MapLoader.

**Further Reading:**

- Online Documentation

## 6.3. Cluster Failover (Enterprise Feature)

As of version 3.12, Hazelcast IMDG Enterprise provides a client failover mechanism that allows for Java client connections to be rerouted to a different cluster without requiring a client network configuration update and client restart. This feature will automatically redirect client traffic to a different cluster during a disaster recovery scenario, and can also be used to manually redirect client traffic in order to perform maintenance or software updates.

**Further Reading:**

- Blue-Green Deployment and Disaster Recovery Documentation

## 6.4. Hot Restart Store (Enterprise HD Feature)

As of version 3.6, Hazelcast IMDG Enterprise HD provides an optional disk-based data-persistence mechanism to enable Hot Restart. This is especially useful when loading cache data from the canonical data source is slow or resource-intensive.



The persistence capability supporting the Hot Restart capability is meant to facilitate cluster restart. It is not intended or recommended for canonical data storage.

With Hot Restart enabled, each member writes its data to local disk using a log-structured persistence algorithm to reduce write latency. A garbage collection thread runs continuously to remove stale data.

### 6.4.1. Hot Restart from Planned Shutdown

Hot Restart Store may be used after either a full-cluster shutdown or member-by-member in a rolling-restart. In both cases, care must be taken to transition the whole cluster or individual cluster members from an “ACTIVE” state to an appropriate inactive state ("PASSIVE" or "FROZEN") to ensure data integrity. (See the documentation on managing cluster and member states for more information on the operating profile of each state.) After this state change, the cluster will freeze the partition table. No new member will be accepted to the cluster and only the existing members will be able to join. After changing the state, you can either perform the rolling restart or a full cluster restart. In either case, the members will load the Hot Restart data. After the procedure is finished, you can set the cluster state to "ACTIVE" again.

### 6.4.2. Hot Restart from Full-Cluster Shutdown

To stop and start an entire cluster using Hot Restart Store, the entire cluster must first be transitioned from an “ACTIVE” state to “PASSIVE” or “FROZEN” prior to shutdown. Full-cluster shutdown may be initiated in any of the following ways:

- Programmatically call the method `HazelcastInstance.getCluster().shutdown()`. This will shut down the entire cluster, automatically causing the appropriate cluster state transitions.
- Change the cluster state from “ACTIVE” to “PASSIVE” or “FROZEN” either programmatically (via `changeClusterState()`) or manually (see the documentation on managing Hot Restart via Management Center); then, manually shut down each cluster member.

### 6.4.3. Hot Restart of Individual Members

Individual members may be stopped and restarted using Hot Restart Store during, for example, a rolling upgrade. Prior to shutdown of any member, the whole cluster must be transitioned from an “ACTIVE” state to “PASSIVE” or “FROZEN”. Once the cluster has safely transitioned to the appropriate state, each member may then be shut down independently. When a member restarts, it will reload its data from disk and re-join the running cluster. When all members have been restarted and joined the cluster, the cluster may be transitioned back to the “ACTIVE” state.



As of version 3.12, members stopped and restarted while the cluster is in an “ACTIVE” state will have their hot restart data automatically removed on startup. This behavior can be changed by setting the `auto-remove-stale-data` property to `false` in the `hot-restart-persistence` section of the Hazelcast configuration.

### 6.4.4. Hot Restart from Unplanned Shutdown

Should an entire cluster crash at once (due, for example, to power or network service interruption), the cluster may be restarted using Hot Restart Store. Each member will attempt to restart using the last saved data: you can start the members at the same time or consecutively and they will load their Hot Restart data. The members wait for the other expected members (the ones present before the crash) to join for `validation-timeout-seconds`. If the expected members don’t join during this period, the cluster starts with the partial data.

There are some edge cases where the last saved state may be unusable, for example, if the cluster

crashes during an ongoing partition migration. In such cases, Hot Restart from local persistence is not possible.

For more information on Hot Restart, see the documentation here.

### 6.4.5. Force Start with Hot Restart Enabled

A member can crash permanently and be unable to recover from the failure. In that case, the restart process cannot be completed since some of the members will not start, or will fail to load their own data. In that case, you can force the cluster to clean its persisted data and make a fresh start. This process is called a Force Start. (See the documentation on Force Start with Hot Restart enabled.)

### 6.4.6. Partial Start with Hot Restart Enabled

When one or more members fail to start, or have incorrect Hot Restart data (stale or corrupted data), or fail to load their Hot Restart data, the cluster will become incomplete and the restart mechanism cannot proceed. One solution is to use Force Start and make a fresh start with existing members. Another solution is to perform a partial start.

A partial start means that the cluster will start with an incomplete member set. Data belonging to those missing members will be assumed lost and Hazelcast IMDG will try to recover missing data using the restored backups. For example, if you have a minimum of two backups configured for all maps and caches, then a partial start with up to two missing members will be safe against data loss. If there are more than two missing members or there are maps/caches with fewer than two backups, then data loss is expected. (See the documentation on partial start with Hot Restart enabled.)

### 6.4.7. Moving/Copying Hot Restart Data

After a Hazelcast IMDG member owning the Hot Restart data is shutdown, the Hot Restart base-dir can be copied/moved to a different server (which may have a different IP address and/or a different number of CPU cores) and the Hazelcast IMDG member can be restarted using the existing Hot Restart data on that new server. Having a new IP address does not affect Hot Restart since it does not rely on the IP address of the server but instead uses Member UUID as a unique identifier. (See the documentation on moving or copying Hot Restart data.)

### 6.4.8. Configuring Hot Restart Store on Intel® Optane™ DC Persistent Memory

Hazelcast can be configured to use Intel® Optane™ DC Persistent Memory (will be mentioned as “Persistent Memory” later in this section from now on) as a Hot Restart directory. For this, you need to perform the following steps:

1. Configure the Persistent Memory as a File System
2. Configure the Hot Restart Store to Use Persistent Memory

Using this Persistent Memory, you can get the Hot Restart times approximately by 250%. Let’s describe the steps in detail in the following sections.

### Configuring the Persistent Memory as a File System

If the persistent memory DIMMs (dual in-line memory modules) are already configured and mounted as a file system, you can skip the instructions given in this section and directly go to the next section.

The persistent memory DIMMs can operate in two modes: MemoryMode or AppDirect. See here for their descriptions. To be able to use it with Hot Restart, DIMMs should be configured with AppDirect mode so you can mount DIMMs as a file system.

The following configuration tools must be installed on your system:

- ipmctl (See https://github.com/intel/ipmctl)
- ndctl (See https://docs.pmem.io/getting-started-guide/installing-ndctl)

The following are the steps:

1. First, check the current setup of the system:

```
[root@localhost builder]# ipmctl show -socket

 SocketID | MappedMemoryLimit | TotalMappedMemory
==================================================
 0x0000   | 4096.0 GiB        | 95.0 GiB
 0x0001   | 4096.0 GiB        | 852.0 GiB
```

The output shown above provides the CPU sockets of the system. You can print the DIMMs of each socket by using its ID, as shown below.

```
[root@localhost builder]# ipmctl show -dimm -socket 0x0000

 DimmID | Capacity  | HealthState | ActionRequired | LockState | FWVersion
==============================================================================
 0x0011 | 126.4 GiB | Healthy     | 0              | Disabled  | 01.00.00.4877
 0x0021 | 126.4 GiB | Healthy     | 0              | Disabled  | 01.00.00.4877
 0x0001 | 126.4 GiB | Healthy     | 0              | Disabled  | 01.00.00.4877
 0x0111 | 126.4 GiB | Healthy     | 0              | Disabled  | 01.00.00.4877
 0x0121 | 126.4 GiB | Healthy     | 0              | Disabled  | 01.00.00.4877
 0x0101 | 126.4 GiB | Healthy     | 0              | Disabled  | 01.00.00.4877
```

You can also see the current configuration of the system, as shown below:

```
[root@localhost builder]# ipmctl show -region

 SocketID | ISetID             | PersistentMemoryType | Capacity  | FreeCapacity |
HealthState
=============================================================================================
=========
 0x0001   | 0xb5b67f48a7c32ccc | AppDirect            | 756.0 GiB | 0.0 GiB      |
Healthy
```

The above example output shows that the DIMMs of the socket with the SocketID `0x0000` is not in use. So, let's configure `0x0000` for Hot Restart following the steps below.

1. Use the following command for the socket `0x0000`:

```
[root@localhost builder]# ipmctl create -goal -socket 0x0000 PersistentMemoryType
=AppDirect

The following configuration will be applied:
 SocketID | DimmID | MemorySize | AppDirect1Size | AppDirect2Size
==========================================================================
 0x0000   | 0x0011 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0021 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0001 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0111 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0121 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0101 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
Do you want to continue? [y/n] y

Created following region configuration goal
 SocketID | DimmID | MemorySize | AppDirect1Size | AppDirect2Size
==========================================================================
 0x0000   | 0x0011 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0021 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0001 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0111 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0121 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
 0x0000   | 0x0101 | 0.0 GiB    | 126.0 GiB      | 0.0 GiB
A reboot is required to process new memory allocation goals.
```

1. Reboot your system. After the reboot, check the regions and namespaces in the system as shown below:

```
[root@localhost builder]# ndctl list --regions --human -N
[
  {
    "dev":"region1",
    "size":"756.00 GiB (811.75 GB)",
    "available_size":0,
    "max_available_extent":0,
    "type":"pmem",
    "iset_id":"0xb5b67f48a7c32ccc",
    "persistence_domain":"memory_controller",
    "namespaces":[
      {
        "dev":"namespace1.0",
        "mode":"fsdax",
        "map":"dev",
        "size":"744.19 GiB (799.06 GB)",
        "uuid":"65121d0e-a8a0-40f1-aed5-8a8ada13b6c7",
        "blockdev":"pmem1"
      }
    ]
  },
  {
    "dev":"region0",
    "size":"756.00 GiB (811.75 GB)",
    "available_size":"756.00 GiB (811.75 GB)",
    "max_available_extent":"756.00 GiB (811.75 GB)",
    "type":"pmem",
    "iset_id":"0x63f47f485dd02ccc",
    "persistence_domain":"memory_controller"
  }
]
```

You can see “region0” has been created with the DIMMs of the socket (ID = 0x0000) in the above output.

1. Now, create a namespace for “region0” as shown below:

```
[root@localhost builder]# ndctl create-namespace --mode fsdax --region region0
{
  "dev":"namespace0.0",
  "mode":"fsdax",
  "map":"dev",
  "size":"744.19 GiB (799.06 GB)",
  "uuid":"87449768-1cc7-4c1b-b138-ea79bc4ee68e",
  "raw_uuid":"6756ef99-744f-4467-90f7-591c0ae162ec",
  "sector_size":512,
  "blockdev":"pmem0",
  "numa_node":0
}
```

1. You should be able see the device as shown below:

```
[root@localhost builder]# ll /dev/pmem0
brw-rw----. 1 root disk 259, 0 Mar 4 02:35 /dev/pmem0
```

1. Format the partition with `ext4` file system using the following command:

```
[root@localhost builder]# mkfs.ext4 /dev/pmem0
```

1. Create a mount point and mount the new filesystem to that mount point using the following commands:

```
[root@localhost builder]# mkdir /mnt/pmem0
[root@localhost builder]# mount -o dax /dev/pmem0 /mnt/pmem0
```

#### Configuring the Hot Restart Store to Use Persistent Memory

After you completed the steps explained in the previous section, you can now create a directory under `/mnt/pmem0` and configure Hazelcast to use that as the Hot Restart directory. See the Configuring Hot Restart section in the Hazelcast IMDG Reference Manual to see how to do it.

As an example, let's create a directory named `hot-restart` under `/mnt/pmem0`:

```
[root@localhost builder]# mkdir /mnt/pmem0/hot-restart
```

To use this as the Hot Restart directory, the configuration should look as follows:

```
<hot-restart-persistence enabled="true">
    <base-dir>/mnt/pmem0/hot-restart</base-dir>
    <parallelism>12</parallelism>
</hot-restart-persistence>
```

You can set `parallelism` to `8` or `12` for the best performance.

### 6.4.9. Hot Backup

During Hot Restart operations you can take a snapshot of the Hot Restart Store at a certain point in time. This is useful when you wish to bring up a new cluster with the same data or parts of the data. The new cluster can then be used to share load with the original cluster, to perform testing/QA, or reproduce an issue using production data.

Simple file copying of a currently running cluster does not suffice and can produce inconsistent snapshots with problems such as resurrection of deleted values or missing values. (See the documentation on hot backup.)

## 6.5. Cluster Scaling: Joining and Leaving Nodes

The oldest node in the cluster is responsible for managing a partition table that maps the ownership of Hazelcast IMDG's data partitions to the nodes in the cluster. When the topology of the cluster changes, such as when a node joins or leaves the cluster, the oldest node rebalances the partitions across the extant nodes to ensure equitable distribution of data. It then initiates the process of moving partitions according to the new partition table. While a partition is in transit to its new node, only requests for data in that partition will block. By default, partition data is migrated in fragments in order to reduce memory and network utilization. This can be controlled using the system property `hazelcast.partition.migration.fragments.enabled`. When a node leaves the cluster, the nodes that hold the backups of the partitions held by the exiting node promote those backup partitions to be primary partitions and are immediately available for access. To avoid data loss, it is important to ensure that all the data in the cluster has been backed up again before taking down other nodes. To shutdown a node gracefully, call the `HazelcastInstance.shutdown()` method, which will block until there is no active data migration and at least one backup of that node's partitions is synced with the new primary ones. To ensure that the entire cluster (rather than just a single node) is in a "safe" state, you may call `PartitionService.isClusterSafe()`. If `PartitionService.isClusterSafe()` returns true, it is safe to take down another node. You may also use the Management Center to determine if the cluster, or a given node, is in a safe state. See the Management Center section below.

Non-map data structures, such as Lists, Sets, Queues, etc., are backed up according to their backup count configuration, but their data is not distributed across multiple nodes. If a node with a non-map data structure leaves the cluster, its backup node will become the primary for that data structure, and it will be backed up to another node. Because the partition map changes when nodes join and leave the cluster, be sure not to store object data to a local filesystem if you persist objects via `MapStore` and `MapLoader` interfaces. The partitions that a particular node is responsible for will almost certainly change over time, rendering locally persisted data inaccessible when the partition table changes.

Starting with 3.9, you have increased control over the lifecycle of nodes joining and leaving by means of a new cluster state `NO_MIGRATION`. In this state, partition rebalancing via migrations and backup replications are not allowed. When performing a planned or unplanned node shutdown you can postpone the actual migration process until the node has rejoined the cluster. This can be useful in the case of large partitions by avoiding a migration both when the node is shutdown and again when it is started.

**Further Reading:**

- Online Documentation, Data Partitioning
- Online Documentation, Partition Service
- Online Documentation, FAQ: How do I know it is safe to kill the second member?
- Online Documentation, Cluster States

# 6.6. Health Check of Hazelcast IMDG Nodes

Hazelcast IMDG provides the HTTP-based Health Check endpoint and the Health Check script.

## 6.6.1. HTTP Health Check

To enable the health check, set the `hazelcast.http.healthcheck.enabled` system property to `true`. By default, it is `false`.

Now, you can retrieve information about your cluster's health status (member state, cluster state, cluster size, etc.) by launching `http://<your member's host IP>:5701/hazelcast/health`.

An example output is given below:

```
Hazelcast::NodeState=ACTIVE
Hazelcast::ClusterState=ACTIVE
Hazelcast::ClusterSafe=TRUE
Hazelcast::MigrationQueueSize=0
Hazelcast::ClusterSize=2
```

## 6.6.2. Health Check script

The `healthcheck.sh` script internally uses the HTTP-based Health endpoint and that is why you also need to set the `hazelcast.http.healthcheck.enabled` system property to `true`.

You can use the script to check Health parameters in the following manner:

```
$ ./healthcheck.sh <parameters>
```

The following parameters can be used:

- `-o`, `--operation`: Health check operation. Available operations:

    - `all`
    - `node-state`
    - `cluster-state`
    - `cluster-safe`
    - `migration-queue-size`
    - `cluster-size`
- `-a`, `--address`: Defines which IP address the Hazelcast member is running on. Default value is `127.0.0.1`.
- `-p`, `--port`: Defines which port the Hazelcast member is running on. Default value is `5701`.

**Example 1: Check Node State of a Healthy Cluster**

Assuming the node is deployed under the address: `127.0.0.1:5701` and it's in the healthy state, the following output is expected.

```
$ ./healthcheck.sh -a 127.0.0.1 -p 5701 -o node-state
ACTIVE
```

**Example 2: Check Cluster Safe of a Non-Existing Cluster**

Assuming there is no node running under the address: `127.0.0.1:5701`, the following output is expected.

```
$ ./healthcheck.sh -a 127.0.0.1 -p 5701 -o cluster-safe
Error while checking health of hazelcast cluster on ip 127.0.0.1 on port 5701.
Please check that cluster is running and that health check is enabled (property set to
true: 'hazelcast.http.healthcheck.enabled' or 'hazelcast.rest.enabled').
```

## 6.7. Shutting Down Hazelcast IMDG Nodes

Ways of shutting down a Hazelcast IMDG node include:

- You can call `kill -9 <PID>` in the terminal (which sends a SIGKILL signal). This will result in an immediate shutdown, which is not recommended for production systems. If you set the property hazelcast.shutdownhook.enabled to `false` and then kill the process using `kill -15 <PID>`, the result is the same (immediate shutdown).
- You can call `kill -15 <PID>` in the terminal (which sends a SIGTERM signal), or you can call the method `HazelcastInstance.getLifecycleService().terminate()` programmatically, or you can use the script `stop.sh` located in your Hazelcast IMDG's `/bin` directory. All three of them will terminate your node ungracefully. They do not wait for migration operations; they force the shutdown. This is much better than `kill -9 <PID>` since it releases most of the used resources.
- In order to gracefully shutdown a Hazelcast IMDG node (so that it waits for the migration operations to be completed), you have four options:
    - You can call the method `HazelcastInstance.shutdown()` programmatically.

- You can use the JMX API's `shutdown` method. You can do this by implementing a JMX client application or using a JMX monitoring tool (like JConsole).
- You can set the property `hazelcast.shutdownhook.policy` to `GRACEFUL` and then shutdown by using `kill -15 <PID>`. Your member will be gracefully shutdown.
- You can use the "Shutdown Member" button in the member view of Hazelcast Management Center.

If you use systemd's `systemctl` utility, i.e., `systemctl stop service_name`, a SIGTERM signal is sent. After 90 seconds of waiting it is followed by a SIGKILL signal by default. Thus, it will call terminate at first, and kill the member directly after 90 seconds. We do not recommend using it with its defaults, but systemd is very customizable and well-documented, and you can see its details using the command `man systemd.kill`. If you can customize it to shutdown your Hazelcast IMDG member gracefully (by using the methods above), then you can use it.

## 6.8. Maintenance and Software Updates

Most software updates and hardware maintenance can be performed without incurring downtime. When removing a cluster member from service, it is important to remember that the remaining members will become responsible for an increased workload. Sufficient memory and CPU headroom will allow for smooth operations to continue. There are four types of updates:

1. Hardware, operating system, or JVM updates. All of these may be updated live on a running cluster without scheduling a maintenance window.



Hazelcast IMDG supports Java versions 6-11 (see the compatibility matrix in Supported JVMs). While not a best practice, JVMs of any supported Java version may be freely mixed and matched between the cluster and its clients, and between individual members of a cluster.

1. Live updates to user application code that executes only on the client side. These updates may be performed against a live cluster with no downtime. Even if the new client-side user code defines new Hazelcast IMDG data structures, these are automatically created in the cluster. As other clients are upgraded they will be able to use these new structures.

Changes to classes that define existing objects stored in Hazelcast IMDG are subject to some restrictions. Adding new fields to classes of existing objects is always allowed. However, removing fields or changing the type of a field will require special consideration. See the section on object schema changes, below.

1. Live updates to user application code that executes on cluster members and on cluster clients. Clients may be updated and restarted without any interruption to cluster operation.
2. Updates to Hazelcast IMDG libraries. Prior to Hazelcast IMDG 3.6, all members and clients of a running cluster had to run the same major and minor version of Hazelcast IMDG. Patch-level upgrades are guaranteed to work with each other. More information is included in the Hazelcast IMDG Software Updates section below.

### 6.8.1. Live Updates to Cluster Member Nodes

In most cases, maintenance and updates may be performed on a running cluster without incurring downtime. However, when performing a live update, you must take certain precautions to ensure the continuous availability of the cluster and the safety of its data.

When you remove a node from service, its data backups on other nodes become active, and the cluster automatically creates new backups and rebalances data across the new cluster topology. Before stopping another member node, you must ensure that the cluster has been fully backed up and is once again in a safe, high-availability state.

The following steps will ensure cluster data safety and high availability when performing maintenance or software updates:

1. Remove one member node from service. You may either kill the JVM process, call `HazelcastInstance.shutdown()`, or use the Management Center.



When you stop a member, all locks and semaphore permits held by that member will be released.

1. Perform the required maintenance or updates on that node's host.
2. Restart the node. The cluster will once again automatically rebalance its data based on the new cluster topology.
3. Wait until the cluster has returned to a safe state before removing any more nodes from service. The cluster is in a safe state when all of its members are in a safe state. A member is in a safe state when all of its data has been backed up to other nodes according to the backup count. You may call `HazelcastInstance.getPartitionService().isClusterSafe()` to determine whether the entire cluster is in a safe state. You may also call `HazelcastInstance.getPartitionService().isMemberSafe(Member member)` to determine whether a particular node is in a safe state. Likewise, the Management Center displays the current safety of the cluster on its dashboard.
4. Continue this process for all remaining member nodes.

### 6.8.2. Live Updates to Clients

A client is a process that is connected to a Hazelcast IMDG cluster with either Hazelcast IMDG's client library (Java, C++, C#, .Net), REST, or Memcached interfaces. Restarting clients has no effect on the state of the cluster or its members, so they may be taken out of service for maintenance or updates at any time and in any order. However, any locks or semaphore permits acquired by a client instance will be automatically released. In order to stop a client JVM, you may kill the JVM process or call HazelcastClient.shutdown().

### 6.8.3. Live Updates to User Application Code that Executes on Both Clients and Cluster Members

Live updates to user application code on cluster members nodes is supported where:

- Existing class definitions do not change (i.e., you are only adding new classes definitions, not

changing existing ones)

- The same Hazelcast IMDG version is used on all members and clients

Examples of what is allowed are new EntryProcessors, ExecutorService, Runnable, Callable, Map/Reduce and Predicates. Because the same code must be present on both clients and members, you should ensure the code is installed on all of the cluster members before invoking that code from a client. As a result, all cluster members must be updated before any client is.

Procedure:

1. Remove one member node from service
2. Update the user libraries on the member node
3. Restart the member node
4. Wait until the cluster is in a safe state before removing any more nodes from service
5. Continue this process for all remaining member nodes
6. Update clients in any order

### 6.8.4. Object Schema Changes

When you release new versions of user code that uses Hazelcast IMDG data, take care to ensure that the object schema for that data in the new application code is compatible with the existing object data in Hazelcast IMDG, or implement custom deserialization code to convert the old schema into the new schema. Hazelcast IMDG supports a number of different serialization methods, one of which, the Portable interface, directly supports the use of multiple versions of the same class in different class loaders. See below for more information on different serialization options.

If you are using object persistence via MapStore and MapLoader implementations, be sure to handle object schema changes there, as well. Depending on the scope of object schema changes in user code updates, it may be advisable to schedule a maintenance window to perform those updates. This will avoid unexpected problems with deserialization errors associated with updating against a live cluster.

## 6.9. Hazelcast IMDG Software Updates

Prior to Hazelcast IMDG version 3.6, all members and clients needed to run the same major and minor version of Hazelcast IMDG. Different patch-level updates are guaranteed to work with each other. For example, Hazelcast IMDG version 3.4.0 will work with 3.4.1 and 3.4.2, allowing for live updates of those versions against a running cluster.

### 6.9.1. Live Updates of Hazelcast IMDG Libraries on Clients

Starting with version 3.6, Hazelcast IMDG supports updating clients with different minor versions. For example, Hazelcast IMDG 3.6.x clients will work with Hazelcast IMDG version 3.7.x.

Where compatibility is guaranteed, the procedure for updating Hazelcast IMDG libraries on clients is as follows:

1. Take any number of clients out of service
2. Update the Hazelcast IMDG libraries on each client
3. Restart each client
4. Continue this process until all clients are updated

### 6.9.2. Updates to Hazelcast IMDG Libraries on Cluster Members

Between Hazelcast IMDG version 3.5 and 3.8, minor version updates of cluster members must be performed concurrently, which requires a scheduled maintenance window to bring the cluster down. Only patch-level updates are supported on members of a running cluster (i.e., rolling upgrade).

Rolling upgrades across minor versions is a feature exclusive to Hazelcast IMDG Enterprise. Starting with Hazelcast IMDG Enterprise 3.8, each minor version released will be compatible with the previous one. For example, it is possible to perform a rolling upgrade on a cluster running Hazelcast IMDG Enterprise 3.8 to Hazelcast IMDG Enterprise 3.9.

The compatibility guarantees described above are given in the context of rolling member upgrades and only apply to GA (general availability) releases. It is never advisable to run a cluster with members running on different patch or minor versions for prolonged periods of time.

For patch-level Hazelcast IMDG updates, use the procedure for live updates on member nodes described above.

For major and minor-level Hazelcast IMDG version updates before Hazelcast IMDG 3.8, use the following procedure:

1. Schedule a window for cluster maintenance
2. Start the maintenance window
3. Stop all cluster members
4. Update Hazelcast IMDG libraries on all cluster member hosts
5. Restart all cluster members
6. Return the cluster to service

### 6.9.3. Rolling Member Upgrades (Enterprise Feature)

As stated above, Hazelcast IMDG supports rolling upgrades across minor versions starting with version 3.8. The detailed procedures for rolling member upgrades can be found in the documentation. (See the documentation on Rolling Member Upgrades.)

# 7. Understanding Configuration Overview

Hazelcast IMDG node requires a valid Configuration file to exist in a known location to the node for discovery. In the event that a valid configuration is not found, the node will start with a default configuration. You configuration is vital to the cluster and determines how a cluster is stood up and

run. In most instances a lot of the detailed to optimize and stabilize a cluster will be done through configuration changes. It is important therefore to understand what these configuration files contain, where they reside, how they are loaded and their overall standardization. You can configure Hazelcast using one or mix of the following options:

- Declarative way
- Programmatic way
- Using Hazelcast system properties
- Within the Spring context
- Dynamically adding configuration on a running cluster

In managing your system you should note how you choose to configure your cluster so as to be able to always validate that you are referencing the appropriate configuration file. It is crucial to note for example whether cluster members, or clients are configured programmatically or through system properties.

When you install Hazelcast IMDG 4.0, you see find default configuration files in the /bin folder: It's important to understand that there are default configuration provided for Hazelcast nodes which will run and stand up a server successfully under optimal conditions. In the event of anomalous behavior always makes sure that you check and validate that the application is running with reference to the correct configuration file and it's not defaulting to one the example configuration files listed below.

- hazelcast.xml: Default declarative XML configuration file for Hazelcast.
- hazelcast.yaml: Default YAML configuration file identical to hazelcast.xml in content.
- hazelcast-full-example.xml: Configuration file which includes all Hazelcast configuration elements and attributes with their descriptions.
- hazelcast-full-example.yaml:
- hazelcast-client-full-example.xml:
- hazelcast-client-full-example.yaml:
- hazelcast-client-failover-full-example.xml:
- hazelcast-client-failover-full-example.yaml:

For your Node instances please ensure that you have your valid Hazelcast configuration file passed in at runtime to the node. This is typically done through the startup script and passing in the path the the correct configuration file.

```
-Dhazelcast.config=`*`<path to the hazelcast.xml or hazelcast.yaml>
```

You configuration file should contain the expected configuration for your system. The default configuration files provided are made available for reference, example and for the nodes to run under default conditions.

You can also review the elements of your cluster configuration checking values in the configuration

file, and by validating that it's the same correct configuration value in though ManCenter. These can include cluster environment, cluster members, ip addresses, network configuration, map configuration etc.

## 7.1. Planning for a Stable Configuration Architecture

In your Hazelcast and/or Hazelcast Client declarative configuration, you can use variables to set the values of the elements. This is valid when you set a system property programmatically or you use the command line interface. You can use a variable in the declarative configuration to access the values of the system properties you set. In this way you can separate out the custom values that need to change for each environment and isolate them in their own properties files and keep the scripts generic and valid for all envrionments without needing to change or update the scripts as your migrate to other runtime environments.

For example, see the following command that sets two system properties.

```
-Dcluster.name=dev
Let's get the values of these system properties in the declarative configuration of
Hazelcast, as shown below.
```

In the XML configuration:

```
<hazelcast>
    <cluster-name>${cluster.name}</cluster-name>
</hazelcast>
```

In the YAML configuration:

```
hazelcast:
  cluster:
    name: ${cluster.name}
This also applies to the declarative configuration of Hazelcast Java Client, as shown
below.
```

```
<hazelcast-client>
    <cluster-name>${cluster.name}</cluster-name>
</hazelcast-client>
```

```
hazelcast-client:
  cluster:
    name: ${cluster.name}
```

Using the abstraction and pattern you can extend and grow your configuration changes with low impact of unwanted changes migrating to new environments.

## 7.2. Dynamic Configuration of Hazelcast 4.x

Hazelcast is configured in a declarative or programmatic way as described above and this startup configuration must be completed before starting a Hazelcast member. This configuration cannot be altered at runtime and as such it is a static configuration. Any changes to the configuration file will not be picked up until after a node restart.

However, It is possible to dynamically add configuration for certain data structures at runtime. This can be done by adding a data structure by invoking one of the Config.add*Config methods on the Config object obtained from a running member's HazelcastInstance.getConfig() method in a scripted manner.

A dynamically added data structure configuration is propagated across all cluster members and as such there is a higher probability of issues and failures due to conditions such as timeout and network partition.

It is strongly recommended understanding all the implications of adding dynamic configuration changes as it impact performance and scalability.

For more detailed information on Hazelcast IMGD 4.x Configuration please review Deployment Section in this Guide as well as the following Reference documentation.

**Further Reading:**

- Online Documentation, Hazelcast Configuration : Hazelcast Configuration

# 8. Starting and Stopping Cluster Runbook

A Hazelcast cluster is a network of members that run Hazelcast. Cluster members on successful startup automatically search for each other on a network and join together to form a cluster. This automatic joining takes place with various discovery mechanisms that the cluster members use to find each other.

The following is a sample set of procedures for starting and stopping a Hazelcast IMDG cluster:

## 8.1. Starting and Stopping Hazelcast

There are working example of Hazelcast Scripts are located in the /bin folder of Hazelcast distributed software package. These scripts provide basic functionalities and examples that can be used as a basis for custom scripts to start and stop members and for cluster management.

The following are selected examples for core scripts with usage and descriptions of each script:

## 8.2. Starting Hazelcast

- start.sh / start.bat: Starts a Hazelcast member with default configuration in the working directory.

## 8.3. Stopping Hazelcast

- stop-all.sh / stop-all.bat: Stops the Hazelcast member that was started in the current working directory.

## 8.4. Hazelcast Cluster Script

- cluster.sh: Provides basic functionalities for cluster management, such as getting and changing the cluster state, shutting down the cluster or forcing the cluster to clean its persisted data and make a fresh start. See the Using the Script cluster.sh section to learn the usage of this script.
- If you running in a production environment it is recommended that you should have developed a set of customer scripts functionally parameterized to deploy by simple context switching.



Similar scripts are recommended for starting and stopping your client applications.

## 8.5. Hazelcast CP Subsystem Startup and Stop

1. When using distributed computing features such as IExecutorService, EntryProcessors, CP Subsystem or Aggregators, any change in application logic or in the implementation of the above features must also be installed on member nodes. All the member nodes must be restarted after new code is deployed using the typical cluster re-deployment process:



If the CP Subsystem is enabled, the stop/re-start process becomes a little bit more structured.

a. Shutdown servers by shutting down the follower nodes first and lastly the leader Node.

b. re-start the servers

# 9. Cluster Sizing

A Hazelcast cluster is a network of members that run Hazelcast.

To size the cluster for your use case, you must first be able to answer the following questions:

- What is your expected data size?
- What are your data access patterns?
- What is your read/write ratio?
- Are you doing more key-based lookups or predicates?
- What are your throughput requirements?
- What are your latency requirements?
- What is your fault tolerance and how many backups do you require?
- Are you using WAN Replication?

# 9.1. Sizing Considerations

Once you know the size, access patterns, throughput, latency, and fault tolerance requirements of your application, you can use the following guidelines to help you determine the size of your cluster. Also, if using WAN Replication, the WAN Replication queue sizes need to be taken into consideration for sizing.

## 9.1.1. Memory Headroom

Once you know the size of your working set of data, you can start sizing your memory requirements. When speaking of "data" in Hazelcast IMDG, this includes both active data and backup data for high availability. The total memory footprint will be the size of your active data plus the size of your backup data. If your fault tolerance allows for just a single backup, then each member of the Hazelcast IMDG cluster will contain a 1:1 ratio of active data to backup data for a total memory footprint of two times the active data. If your fault tolerance requires two backups, then that ratio climbs to 1:2 active to backup data for a total memory footprint of three times your active data set. If you use only heap memory, each Hazelcast IMDG node with a 4GB heap should accommodate a maximum of 3.5 GB of total data (active and backup). If you use the High-Density Data Store, up to 75% of your physical memory footprint may be used for active and backup data, with headroom of 25% for normal fragmentation. In both cases, however, the best practice is to keep some memory headroom available to handle any node failure or explicit node shutdown. When a node leaves the cluster, the data previously owned by the newly offline node will be redistributed across the remaining members. For this reason, we recommend that you plan to use only 60% of available memory, with 40% headroom to handle node failure or shutdown.

## 9.1.2. Recommended Configurations

Hazelcast IMDG performs scaling tests for each version of the software. Based on this testing we specify some scaling maximums. These are defined for each version of the software starting with 3.6. We recommend staying below these numbers. Please contact Hazelcast if you plan to use higher limits.

- Maximum 100 multi-socket clients per Member
- Maximum 1,000 unisocket clients per Member
- Maximum of 100GB HD Memory per Member

In the documentation, multi-socket clients are called smart clients. Each client maintains a connection to each member. Unisocket clients have a single connection to the entire cluster.

## 9.1.3. Very Low-Latency Requirements

If your application requires very low latency, consider using an embedded deployment. This configuration will deliver the best latency characteristics. Another solution for ultra-low-latency infrastructure could be `ReplicatedMap`. `ReplicatedMap` is a distributed data structure that stores an exact replica of data on each node. This way, all of the data is always present on every node in the cluster, thus preventing a network hop across to other nodes in the case of a `map.get()` request. Otherwise, the isolation and scalability gains of using a client-server deployment are preferable.

### 9.1.4. CPU Sizing

As a rule of thumb, we recommend a minimum of 8 cores per Hazelcast IMDG server instance. You may need more cores if your application is CPU-heavy, for example, in a high throughput distributed executor service deployment.

## 9.2. Example: Sizing a Cache Use Case

Consider an application that uses Hazelcast IMDG as a data cache. The active memory footprint will be the total number of objects in the cache times the average object size. The backup memory footprint will be the active memory footprint times the backup count. The total memory footprint is the active memory footprint plus the backup memory footprint:

```
Total memory footprint = (total objects * average object size) + (total objects *
average object size * backup count)
```

For this example, let's stipulate the following requirements:

- 50 GB of active data
- 40,000 transactions per second
- 70:30 ratio of reads to writes via map lookups
- Less than 500 ms latency per transaction
- A backup count of 2

### 9.2.1. Cluster Size Using the High-Density Memory Store

Since we have 50 GB of active data, our total memory footprint will be:

- 50 GB + 50 GB * 2 (backup count) = 150 GB.

Add 40% memory headroom and you will need a total of 250 GB of RAM for data.

To satisfy this use case, you will need three Hazelcast IMDG nodes, each running a 4 GB heap with ~84 GB of data off-heap in the High-Density Data Store.



You cannot have a backup count greater than or equal to the number of nodes available in the cluster. Hazelcast will ignore higher backup counts and will create the maximum number of backup copies possible. For example, Hazelcast IMDG will only create two backup copies in a cluster of three nodes, even if the backup count is set equal to or higher than three.



No node in a Hazelcast IMDG cluster will store primary as well as its own backup.

### 9.2.2. Cluster Size Using Only Heap Memory

Since it's not practical to run JVMs with greater than a 4 GB heap, you will need a minimum of 42

JVMs, each with a 4 GB heap to store 150 GB of active and backup data as a 4 GB JVM would give approximately 3.5 GB of storage space. Add the 40% headroom discussed earlier, for a total of 250 GB of usable heap, then you will need ~72 JVMs, each running with four GB heap for active and backup data. Considering that each JVM has some memory overhead and Hazelcast's rule of thumb for CPU sizing is eight cores per Hazelcast IMDG server instance, you will need at least 576 cores and upwards of 300 GB of memory.

### 9.2.3. Summary

150 GB of data, including backups.

High-Density Memory Store: * 3 Hazelcast IMDG nodes * 24 cores * 256 GB RAM

Heap-only: * 72 Hazelcast IMDG nodes * 576 cores * 300 GB RAM

# 10. Security & Hardening

Hazelcast IMDG Enterprise offers a rich set of security features you can use:

- Authentication for cluster members and clients
- Access control checks on client operations
- Socket and Security Interceptor
- TLS/SSL
- OpenSSL integration
- Mutual Authentication on TLS/SSL
- Symmetric Encryption
- Secret (group password, symmetric encryption password and salt) validation including strength policy
- Java Authentication and Authorization Services
- FIPS Compliant Mode

## 10.1. Features (Enterprise and Enterprise HD)

The major security features are described below. Please see the Security section of the Hazelcast IMDG Reference Manual for details.

### 10.1.1. Socket Interceptor

The socket interceptor allows you to intercept socket connections before a node joins a cluster or a client connects to a node. This provides the ability to add custom hooks to the cluster join operation and perform connection procedures (like identity checking using Kerberos, etc.).

### 10.1.2. Security Interceptor

The security interceptor allows you to intercept every remote operation executed by the client. This lets you add very flexible custom security logic.

### 10.1.3. Encryption

All socket-level communication among all Hazelcast IMDG members can be encrypted. Encryption is based on the Java Cryptography Architecture.

### 10.1.4. TLS/SSL

All Hazelcast IMDG members can use TLS socket communication among each other.

### 10.1.5. OpenSSL Integration

TLS/SSL in Java is normally provided by the JRE. However, the performance overhead can be significant on some Java versions. On most popular OS distributions (Linux, Mac OS, Windows), you can leverage OpenSSL/BoringSSL integration provided by Hazelcast IMDG, which brings significant performance improvements in some cases.

### 10.1.6. Mutual Authentication on SSL/TLS

TLS mutual authentication is supported. This allows the clients to have their keyStores, and members to have their trustStores, so that the members can know which clients they can trust.

### 10.1.7. TLS Certificate updates

It's possible to change TLS certificates in the running Hazelcast cluster with zero down-time. The certificates can be updated on members one by one. See Hazelcast Reference manual section *Updating Certificates in the Running Cluster* for details.

### 10.1.8. Credentials and ClusterLoginModule

The Credentials interface and ClusterLoginModule allow you to implement custom credentials checking. The default implementation that comes with Hazelcast IMDG uses a username/password scheme.

Note that cluster passwords are stored as clear text inside the hazelcast.xml or hazelcast.yaml configuration file. This is the default behavior, and if someone has access to read the configuration file, then they can join a node to the cluster. However, you can easily provide your own credentials factory by using the CredentialsFactoryConfig API and then setting up the LoginModuleConfig API to handle the joins to the cluster.

### 10.1.9. Cluster Member Security

Hazelcast IMDG Enterprise supports standard Java Security (JAAS) based authentication between cluster members.

### 10.1.10. Native Client Security

Hazelcast IMDG's client security includes both authentication and authorization via configurable permissions policies.

**Further Reading:**

- Online Documentation

### 10.1.11. FIPS Compliant Mode

As of version 3.12, Hazelcast IMDG Enterprise can run in FIPS compliant mode, i.e. the underlying system uses FIPS validated cryptographic modules.

**Further Reading:** http://docs.hazelcast.org/docs/latest/manual/html-single/index.html#fips-140-2

## 10.2. Validating Secrets Using Strength Policy

Hazelcast IMDG Enterprise offers a secret validation mechanism including a strength policy. The term "secret" here refers to the cluster group password, symmetric encryption password and salt, and other passwords and keys.

For this validation, Hazelcast IMDG Enterprise comes with the class DefaultSecretStrengthPolicy to identify all possible weaknesses of secrets and to display a warning in the system logger. Note that, by default, no matter how weak the secrets are, the cluster members will still start after logging this warning; however, this is configurable (please see the "Enforcing the Secret Strength Policy" section).

Requirements (rules) for the secrets are as follows: * Minimum length of eight characters; and * Large keyspace use, ensuring the use of at least three of the following: **mixed case;** alpha; **numerals;** special characters; and ** no dictionary words.

The rules "Minimum length of eight characters" and "no dictionary words" can be configured using the following system properties:

- `hazelcast.security.secret.policy.min.length`: Set the minimum secret length. The default is 8 characters. Example: -Dhazelcast.security.secret.policy.min.length=10
- `hazelcast.security.dictionary.policy.wordlist.path`: Set the path of a wordlist available in the file system. The default is /usr/share/dict/words. Example: -Dhazelcast.security.dictionary.policy.wordlist.path="/Desktop/myWordList"

### 10.2.1. Using a Custom Secret Strength Policy

You can implement SecretStrengthPolicy to develop your custom strength policy for a more flexible or strict security. After you implement it, you can use the following system property to point to your custom class:

`hazelcast.security.secret.strength.default.policy.class`: Set the full name of the custom class.

Example: `-Dhazelcast.security.secret.strength.default.policy.class="com.impl.myStrengthPolicy"`

### 10.2.2. Enforcing the Secret Strength Policy

By default, secret strength policy is NOT enforced. This means that if a weak secret is detected, an informative warning will be shown in the system logger and the members will continue to initialize. However, you can enforce the policy using the following system property so that the members will not be started until the weak secret errors are fixed:

`hazelcast.security.secret.strength.policy.enforced`: Set to "true" to enforce the secret strength policy. The default is "false". To enforce:

```
-Dhazelcast.security.secret.strength.policy.enforced=true
```

The following is a sample warning when secret strength policy is NOT enforced, i.e., the above system property is set to "false":

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ SECURITY WARNING @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Group password does not meet the current policy and complexity requirements.
*Must not be set to the default.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

The following is a sample warning when secret strength policy is enforced, i.e., the
above system property is set to “true”:

WARNING: [192.168.2.112]:5701 [dev] [3.9-SNAPSHOT]
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ SECURITY WARNING @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Symmetric Encryption Password does not meet the current policy and complexity
requirements.
*Must contain at least 1 number.
*Must contain at least 1 special character.
Group Password does not meet the current policy and complexity requirements.
*Must not be set to the default.
*Must have at least 1 lower and 1 uppercase characters.
*Must contain at least 1 number.
*Must contain at least 1 special character.
Symmetric Encryption Salt does not meet the current policy and complexity
requirements.
*Must contain 8 or more characters.
*Must contain at least 1 number.
*Must contain at least 1 special character.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Exception in thread "main" com.hazelcast.security.WeakSecretException: Weak secrets
found in configuration, check output above for more details.
at com.hazelcast.security.impl.WeakSecretsConfigChecker.evaluateAndReport
(WeakSecretsConfigChecker.java:49)
at com.hazelcast.instance.EnterpriseNodeExtension.printNodeInfo
(EnterpriseNodeExtension.java:197)
at com.hazelcast.instance.Node.<init>(Node.java:194)
at com.hazelcast.instance.HazelcastInstanceImpl.createNode(HazelcastInstanceImpl.java
:163)
at com.hazelcast.instance.HazelcastInstanceImpl.<init>(HazelcastInstanceImpl.java:130)
at com.hazelcast.instance.HazelcastInstanceFactory.constructHazelcastInstance
(HazelcastInstanceFactory.java:195)
at com.hazelcast.instance.HazelcastInstanceFactory.newHazelcastInstance
(HazelcastInstanceFactory.java:174)
at com.hazelcast.instance.HazelcastInstanceFactory.newHazelcastInstance
(HazelcastInstanceFactory.java:124)
at com.hazelcast.core.Hazelcast.newHazelcastInstance(Hazelcast.java:58)
```

## 10.3. Security Defaults

Hazelcast IMDG port 5701 is used for all communication by default. Please see the port section in the Reference Manual for different configuration methods, and its attributes.

- REST is disabled by default.

- Memcache is disabled by default.

## 10.4. Advanced networking

Starting with Hazelcast IMDG 3.12, it is possible to configure the Hazelcast members with separate server sockets using a different network configuration for different protocols. This configuration scheme allows better security control as it gives users possibility to bind the supported protocols on different network sockets and network interfaces. The protocol endpoints can have different TLS configuration.

Enpoints available for explicit advanced configuration:

- Hazelcast Member Protocol
- Hazelcast Open Binary Client Protocol
- Hazelcast WAN Endpoints (outgoing, incoming)
- HTTP REST API
- Memcache Text Protocol

The following is a sample configuration where member and client endpoints are enabled on different ports. The client protocol is protected by TLS with mutual authentication.

```
<hazelcast>
   <advanced-network enabled="true">
       <join>
           <multicast enabled="false"/>
           <tcp-ip enabled="true">
               <member>10.0.0.1</member>
               <member>10.0.0.2</member>
               <member>10.0.0.3</member>
           </tcp-ip>
       </join>
       <member-server-socket-endpoint-config>
           <port>5701</port>
        </member-server-socket-endpoint-config>
        <client-server-socket-endpoint-config>
            <port>9090</port>
            <ssl enabled="true">
                <properties>
                    <property name="keyStore">/opt/clientsocket-
keystore.jks</property>
                    <property name="keyStorePassword">secret.123</property>
                    <property name="trustStore">/opt/clientsocket-
truststore.jks</property>
                    <property name="trustStorePassword">changeit</property>
                    <property name="protocol">TLSv1.2</property>
                    <property name="mutualAuthentication">REQUIRED</property>
                </properties>
            </ssl>
        </client-server-socket-endpoint-config>
    </advanced-network>
</hazelcast>
```

Hazelcast network interfaces configuration and/or system firewall rules can be used to restrict access to the member endpoint from untrusted networks.

**Further Reading:** https://docs.hazelcast.org/docs/latest/manual/html-single/index.html#advanced-network-configuration

## 10.5. Hardening Recommendations

For enhanced security, we recommend the following hardening recommendations:

- Hazelcast IMDG members, clients or Management Center should not be deployed internet-facing or on non-secure networks or non-secure hosts.
  - If clients need to access the the cluster from an untrusted network, use the advanced-networking feature to only provide access to the client protocol.
- Any unused port, except the Hazelcast IMDG port (default 5701), should be closed.
- If Memcache is not used, ensure Memcache is not enabled (disabled by default):

    - Related system property is hazelcast.memcache.enabled
    - Please see the System Properties section in the Hazelcast IMDG Reference Manual for more information.
- If REST is not used, ensure REST is not enabled (disabled by default)
    - Please see the Using the REST Endpoint Groups section in the Hazelcast IMDG Reference Manual for more information.
- Configuration variables can be used in declarative mode to access the values of the system properties you set.
    - For example, see the following command that sets two system properties: -Dgroup.name=dev -Dgroup.password=somepassword
    - Please see the Using Variables section in the Hazelcast IMDG Reference Manual for more information.
- Starting with Hazelcast IMDG 3.9.4, variable replacers can be used to replace custom strings during loading the configuration.
    - For example, they can be used to mask sensitive information such as usernames and passwords. However, their usage is not limited to security-related information.
    - Please see Variable Replacers section in the Hazelcast IMDG Reference Manual for more information about usage and examples.
- Restrict the users and the roles of those users in Management Center. The “Administrator role” in particular is a super user role that can access the “Scripting” and “Console” tabs of Management Center where they can reach and/or modify cluster data and should be restricted. The Read-Write User role also provides Scripting access which can be used to read or modify values in the cluster. Please see administering-management-center for more information.
- By default, Hazelcast IMDG lets the system pick up an ephemeral port during a socket bind operation, but security policies/firewalls may require you to restrict outbound ports to be used by Hazelcast IMDG-enabled applications, including Management Center. To fulfill this requirement, you can configure Hazelcast IMDG to use only defined outbound ports. Please see Outbound Ports for different configuration methods.
- TCP/IP discovery is recommended where possible. Please see discovering-cluster-members for different discovery mechanisms.
- Hazelcast IMDG allows you to intercept every remote operation executed by the client. This lets you add a very flexible custom security logic. Please see security-interceptor for more information.
- Hazelcast IMDG by default transmits data between clients and members, and members and members in plain text. This configuration is not secure. In more secure environments, TLS or symmetric encryption should be enabled. Please see security.
- With Symmetric Encryption, the symmetric password is stored in hazelcast.xml or hazelcast.yaml. Access to these files should be restricted.
- With TLS Security, the keystore is used. The keystore password is in the hazelcast.xml or hazelcast.yaml configuration file, and, if clients are used, also in the hazelcast-client.xml or hazelcast-client.yaml configuration file. Access to these files should be restricted.

- A custom trust store can be used by setting the trustStore path in the SSL configuration, which then avoids using the default trust store.
- We recommend that Mutual TLS Authentication be enabled on a Hazelcast IMDG production cluster.
- Hazelcast IMDG uses Java serialization for some objects transferred over the network. To avoid deserialization of objects from untrusted sources Hazelcast offers some protection mechanisms. We recommend enabling Mutual TLS Authentication and disabling Multicast Join configuration. Hazelcast IMDG 3.11 introduces Java serialization filter configuration and we recommend to use it for whitelisting set of trusted classes or packages which are allowed for deserialization.
- Starting with Hazelcast IMDG 3.11.1, script execution can be disabled on Hazelcast members. Scripts executed from Management center have access to system resources (files etc.) with privileges of user running Hazelcast. We recommend that scripting be disabled on members.

## 10.6. Secure Context

Hazelcast IMDG's security features can be undermined by a weak security context. The following areas are critical:

- host security
- development and test security

### 10.6.1. Host Security

Hazelcast IMDG does not encrypt data held in memory since it is "data in use," NOT "data at rest." Similarly, the Hot Restart Store does not encrypt data. Finally, encryption passwords or Java keystore passwords are stored in hazelcast.xml or hazelcast.yaml and hazelcast-client.xml or hazelcast-client.yaml, which are on the file system. Management Center passwords are also stored on the Management Center host.

An attacker with host access to either a Hazelcast IMDG member host or a Hazelcast IMDG client host with sufficient permission could, therefore, read data held either in memory or on disk and be in a position to obtain the key repository, though perhaps not the keys themselves.

Memory contents should be secured by securing the host. Hazelcast IMDG assumes the host is already secured. If there is concern about dumping a process then the value can be encrypted before it is placed in the cache.

### 10.6.2. Development and Test Security

Because encryption passwords or Java keystore passwords are stored in hazelcast.xml or hazelcast.yaml and hazelcast-client.xml or hazelcast-client.yaml, which are on the file system, different passwords should be used for production and for development. Otherwise, the development and test teams will know these passwords.

### 10.6.3. Java Security

Hazelcast IMDG is primarily Java-based. Java is less prone to security problems than C with security

designed in; however, the Java version being used should be immediately patched with any security patches.

# 11. Deployment and Scaling Runbook

The following is a sample set of procedures for deploying and scaling a Hazelcast IMDG cluster:

1. Ensure you have the appropriate Hazelcast IMDG jars (`hazelcast-ee` for Enterprise) installed. Normally `hazelcast-all-<version>.jar` is sufficient for all operations, but you may also install the smaller `hazelcast-<version>.jar` on member nodes and `hazelcast-client-<version>.jar` for clients.

## 11.1. Installation

This sections explains the installation or upgrade of Hazelcast IMDG and Hazelcast IMDG Enterprise.

## 11.2. Pre-Requisites

The only prerequisite for Hazelcast IMDG Enterprise is Java availability and a valid License Key.

## 11.3. Java JDK

Java JDK must be installed on the target system at a location that is available to the Hazelcast Instance. This can be set via an Environment Property or set/passed as a parameter to the Hazelcast startup scripts.

The Following Java Virtual Machines Versions are supported

- Java 13.x.x
    1. Java 12.x.x
    2. Java 11.x.x
    3. Java 9.x.x
    4. Java 8.x.x
    5. Java 7.x.x - (Supported only on certain versions with caveats)
    6. Java 6.x.x - (Supported only on certain versions with caveats)

You can find more details on supported Java Virtual Machines here. * https://docs.hazelcast.org/docs/latest/manual/html-single/index.html#supported-jvms

## 11.4. License on the System

There are 2 modes of providing license details to a Hazelcast IMDG Enterprise Instance. Provide the License key Programmatically .To provide the license key programmatically please review the

following reference documentation on how to implement license key.

**Further Reading:** * License key

Provide the License Key via hazelcast.xml or hazelcast.yaml configuration file.

## 11.5. Hazelcast Configuration file

1. If not configured programmatically, Hazelcast IMDG Server looks for a hazelcast.xml or hazelcast.yaml configuration file.
2. The same goes for Hazelcast Clients. You must configure a license key in a hazelcast-client.xml or hazelcast-client.yaml configuration file for client operations.
3. Place all the configurations at their respective places so that they can be picked by their respective applications (Hazelcast IMDG server or an application client).
    - There are 2 ways to pass the configuration file to the instance.
4. System property: Hazelcast will first check if "hazelcast.config" system property is set to a file path. If it is set, its value is used as the path to the valid hazelcast configuration file. This is useful if you want to be able to change your Hazelcast configuration; you can do this because it is not embedded within the application. You can set the System Property hazelcast.config option and pass it to the application as shown with the following command:

```
-Dhazelcast.config=`*`<path to the hazelcast.xml or hazelcast.yaml>
```

1. Classpath: If the System Property hazelcast.config is not set, Hazelcast will check classpath for a valid hazelcast-<variable>.xml file. If Hazelcast Instance doesn't find any valid Hazelcast config file, it will start with an internal default configuration (hazelcast-default.xml) located in hazelcast.jar.

**Further Reading:** * Understanding Configuration

# 12. Starting and Stopping Hazelcast

At startup members automatically search for each other on a network and join together to form a cluster. This automatic joining takes place with various discovery mechanisms that the cluster members use to find each other.

Hazelcast Scripts to start and stop members are located in the scripts folder scripts under the /bin folder. These scripts provide basic functionalities for member and cluster management.

The following are selected examples for core scripts with usage and descriptions of each script:

- start.sh / start.bat: Starts a Hazelcast member with default configuration in the working directory.
- stop.sh / stop.bat: Stops the Hazelcast member that was started in the current working directory.
- cluster.sh: Provides basic functionalities for cluster management, such as getting and changing

the cluster state, shutting down the cluster or forcing the cluster to clean its persisted data and make a fresh start. See the Using the Script cluster.sh section to learn the usage of this script.

If you running in a production environment it is recommended that you should have developed a set of customer scripts functionally parameterized to deploy by simple context switching.

# 13. Cluster Discovery Basics

1. The recommended minimum number of nodes in a stable cluster is 3 members. Cluster members automatically find each other and join together to form a cluster. This discovery and joining takes place with various discovery mechanisms that the cluster members use to find each other. Make sure that you have provided the IP addresses of a minimum of two Hazelcast IMDG server nodes and the IP address of the joining node itself, if there are more than two nodes in the cluster, in both the configurations. This is required to avoid new nodes failing to join the cluster if the IP address that was configured does not have any server instance running on it.



A Hazelcast IMDG member looks for a running cluster at the IP addresses provided in its configuration. For the upcoming member to join a cluster, it should be able to detect the running cluster on any of the IP addresses provided. The same applies to clients as well.



After a cluster is formed, communication between cluster members is always via TCP/IP, regardless of the discovery mechanism used.

The Basic Typical Discovery Mechanism are TCP and Multicast . Multicast is the default, basic and recommended way to enable Cluster Members to discover each other. Multicast allows Hazelcast cluster members to find each other using multicast communication. As Multicast relies on UPD Multicast mechanism is not recommended for production use since UDP is often blocked in production environments. In this case you can configure Hazelcast to be a full TCP/IP cluster. See the Discovering Members by TCP section for configuration details.

1. Discovering Members by TCP If multicast is not the preferred way of discovery then its recommended to enable discovery via TCP and configure Hazelcast to be a full TCP/IP cluster. When you configure Hazelcast to discover members by TCP/IP, you must list all or a subset of the members' hostnames and/or IP addresses as cluster members. You do not have to list all of these cluster members, but at least one of the listed members has to be active in the cluster when a new member joins.

Cloud Discovery Mechanisms **Further Reading:** * Cluster Discovery

# 14. Smart Clients

1. Enable “smart” routing on clients. This is done to avoid a client sending all of its requests to the cluster routed through a Hazelcast IMDG member, hence bottlenecking that member. A smart client connects with all Hazelcast IMDG server instances and sends all of its requests directly to the respective member node. This improves the latency and throughput of Hazelcast IMDG data

access.

**Further Reading:** * Online Documentation

1. Make sure that all nodes are reachable by every other node in the cluster and are also accessible by clients (ports, network, etc).
2. Start Hazelcast IMDG server instances first. While not mandatory, this is a best practice to avoid clients timing out or complaining that no Hazelcast IMDG server is found which can happen if clients are started before the server.
3. Enable/start a network log collecting utility. `nmon` is perhaps the most commonly used tool and is very easy to deploy.
4. To add more server nodes to an already running cluster, start a server instance with a similar configuration to the other nodes with the possible addition of the IP address of the new node. A maintenance window is not required to add more nodes to an already running Hazelcast IMDG cluster.



When a node is added or removed to a Hazelcast IMDG cluster, clients may see a little pause time, but this is normal. This is essentially the time required by Hazelcast IMDG servers to rebalance the data upon the arrival or departure of a member node.



There is no need to change anything on the clients when adding more server nodes to the running cluster. Clients will update themselves automatically to connect to the new node once it has successfully joined the cluster.



Rebalancing of data (primary plus backup) on arrival or departure (forced or unforced) of a node is an automated process and no manual intervention is required.



You can promote your lite members to become data members. To do this, either use the Cluster API or the Management Center UI.

**Further Reading:** * Online Documentation

1. Check that you have configured an adequate backup count based on your SLAs.
2. When using distributed computing features such as IExecutorService, EntryProcessors, Map/Reduce or Aggregators, any change in application logic or in the implementation of the above features must also be installed on member nodes. All the member nodes must be restarted after new code is deployed using the typical cluster re-deployment process:
    a. Shutdown servers
    b. Deploy the new application jar in the servers' classpath
    c. Start servers

# 15. Hazelcast Cloud Deployment Discovery Runbook

Hazelcast 4.0 now includes options for Cluster deployment of cloud environments using Hazelcast Cloud

There is now the option to deploy Hazelcast using Hazelcast Cloud. Hazelcast Cloud is enterprise-grade integration service for delivering Hazelcast software in the cloud. It allows Operations to deploy, scale and update Hazelcast Instances consistently and efficiently using Hazelcast Cloud as well as provide a platform to maintain the cluster environment. Another use of Hazelcast Cloud is as a low-latency high-performance caching or data layer for your micro-services, as well as a nice solution for state management of server-less functions e.g. (AWS Lambda).

Hazelcast Cloud integrates Docker and Kubernetes, and is powered by Hazelcast IMDG Enterprise HD. Hazelcast Cloud supports deployment of clusters on Amazon Web Services (AWS), Microsoft Azure and as of early December 2020, Google Cloud Platform (GCP). Hazelcast is based on Hazelcast IMDG Enterprise HD, and features advanced functionalities such as TLS, multi-region support, robust, scalable, fast persistence, and high availability.

# 16. Cloud Deployment

Hazelcast provides a number of Cloud deployment options including Docker, Kubernetes/OpenShift, VMware Tanzu (former Pivotal Cloud Foundry), AWS, Azure, GCP, and more. For details please check Online Documentation, Installing and Upgrading.

# 17. Deployment and Scaling Runbook

The following is a sample set of procedures for deploying and scaling a Hazelcast IMDG cluster:

1. Ensure you have the appropriate Hazelcast IMDG jars (`hazelcast-ee` for Enterprise) installed. Normally `hazelcast-all-<version>.jar` is sufficient for all operations, but you may also install the smaller `hazelcast-<version>.jar` on member nodes and `hazelcast-client-<version>.jar` for clients.

# 18. Installation

This sections explains the installation or upgrade of Hazelcast IMDG and Hazelcast IMDG Enterprise.

# 19. Pre-Requisites

The only prerequisite for Hazelcast IMDG Enterprise is Java availability and a valid License Key.

# 20. Java JDK

Java JDK must be installed on the target system at a location that is available to the Hazelcast Instance. This can be set via an Environment Property or set/passed as a parameter to the Hazelcast startup scripts.

The Following Java Virtual Machines Versions are supported

1. Java 13.x.x
2. Java 12.x.x
3. Java 11.x.x
4. Java 9.x.x
5. Java 8.x.x
6. Java 7.x.x - (Supported only on certain versions with caveats)
7. Java 6.x.x - (Supported only on certain versions with caveats)

You can find more details on supported Java Virtual Machines here.

- https://docs.hazelcast.org/docs/latest/manual/html-single/index.html#supported-jvms

# 21. License on the System

There are 2 modes of providing license details to a Hazelcast IMDG Enterprise Instance.

- Provide the License key Programmatically .To provide the license key programmatically please review the following reference documentation on how to implement license key.
- https://docs.hazelcast.org/docs/latest/manual/html-single/index.html#setting-the-license-key

Provide the License Key via hazelcast.xml or hazelcast.yaml configuration file.

# 22. Hazelcast Configuration file

1. If not configured programmatically, Hazelcast IMDG Server looks for a hazelcast.xml or hazelcast.yaml configuration file.
2. The same goes for Hazelcast Clients. You must configure a license key in a hazelcast-client.xml or hazelcast-client.yaml configuration file for client operations.
3. Place all the configurations at their respective places so that they can be picked by their respective applications (Hazelcast IMDG server or an application client).
    - There are 2 ways to pass the configuration file to the instance.
4. System property: Hazelcast will first check if "hazelcast.config" system property is set to a file path. If it is set, its value is used as the path to the valid hazelcast configuration file. This is useful if you want to be able to change your Hazelcast configuration; you can do this because it is not embedded within the application. You can set the System Property hazelcast.config option

and pass it to the application as shown with the following command:

-Dhazelcast.config=*<path to the hazelcast.xml or hazelcast.yaml>

1. Classpath: If the System Property hazelcast.config is not set, Hazelcast will check classpath for a valid hazelcast-<variable>.xml file. If Hazelcast Instance doesn't find any valid Hazelcast config file, it will start with an internal default configuration (hazelcast-default.xml) located in hazelcast.jar.

You can read more about Hazelcast configuration files here: * https://docs.hazelcast.org/docs/latest/manual/html-single/index.html#understanding-configuration

# 23. Starting and Stopping Hazelcast

At startup members automatically search for each other on a network and join together to form a cluster. This automatic joining takes place with various discovery mechanisms that the cluster members use to find each other.

Hazelcast Scripts to start and stop members are located in the scripts folder scripts under the /bin folder. These scripts provide basic functionalities for member and cluster management.

The following are selected examples for core scripts with usage and descriptions of each script:

- start.sh / start.bat: Starts a Hazelcast member with default configuration in the working directory.
- stop.sh / stop.bat: Stops the Hazelcast member that was started in the current working directory.
- cluster.sh: Provides basic functionalities for cluster management, such as getting and changing the cluster state, shutting down the cluster or forcing the cluster to clean its persisted data and make a fresh start. See the Using the Script cluster.sh section to learn the usage of this script.

If you running in a production environment it is recommended that you should have developed a set of customer scripts functionally parameterized to deploy by simple context switching.

# 24. Cluster Discovery Basics

1. The recommended minimum number of nodes in a stable cluster is 3 members. Cluster members automatically find each other and join together to form a cluster. This discovery and joining takes place with various discovery mechanisms that the cluster members use to find each other. Make sure that you have provided the IP addresses of a minimum of two Hazelcast IMDG server nodes and the IP address of the joining node itself, if there are more than two nodes in the cluster, in both the configurations. This is required to avoid new nodes failing to join the cluster if the IP address that was configured does not have any server instance running on it.



A Hazelcast IMDG member looks for a running cluster at the IP addresses provided in its configuration. For the upcoming member to join a cluster, it should be able to detect the running cluster on any of the IP addresses provided. The same applies to clients as well.



After a cluster is formed, communication between cluster members is always via TCP/IP, regardless of the discovery mechanism used.

The Basic Typical Discovery Mechanism are TCP and Multicast . Multicast is the default, basic and recommended way to enable Cluster Members to discover each other. Multicast allows Hazelcast cluster members to find each other using multicast communication. As Multicast relies on UPD Multicast mechanism is not recommended for production use since UDP is often blocked in production environments. In this case you can configure Hazelcast to be a full TCP/IP cluster. See the Discovering Members by TCP section for configuration details.

1. Discovering Members by TCP If multicast is not the preferred way of discovery then it's recommended to enable discovery via TCP and configure Hazelcast to be a full TCP/IP cluster. When you configure Hazelcast to discover members by TCP/IP, you must list all or a subset of the members' hostnames and/or IP addresses as cluster members. You do not have to list all of these cluster members, but at least one of the listed members has to be active in the cluster when a new member joins.

Cloud Discovery Mechanisms **Further Reading:** * Discovery

# 25. Smart Clients

1. Enable "smart" routing on clients. This is done to avoid a client sending all of its requests to the cluster routed through a Hazelcast IMDG member, hence bottlenecking that member. A smart client connects with all Hazelcast IMDG server instances and sends all of its requests directly to the respective member node. This improves the latency and throughput of Hazelcast IMDG data access.

**Further Reading:** * Online Documentation

# 26. Cluster Audit and Monitoring

This sections outlines basic steps for monitoring and auditing your cluster. It outlines basic health checks for your cluster environment. You should also review ManCenter documentation on how to Monitor and audit your cluster.

1. Make sure that all nodes are reachable by every other node in the cluster and are also accessible by clients (ports, network, etc).
2. Start Hazelcast IMDG server instances first. While not mandatory, this is a best practice to avoid clients timing out or complaining that no Hazelcast IMDG server is found which can happen if clients are started before the server.
3. Enable/start a network log collecting utility. `nmon` is perhaps the most commonly used tool and is

very easy to deploy.

4. To add more server nodes to an already running cluster, start a server instance with a similar configuration to the other nodes with the possible addition of the IP address of the new node. A maintenance window is not required to add more nodes to an already running Hazelcast IMDG cluster.



When a node is added or removed to a Hazelcast IMDG cluster, clients may see a little pause time, but this is normal. This is essentially the time required by Hazelcast IMDG servers to rebalance the data upon the arrival or departure of a member node.



There is no need to change anything on the clients when adding more server nodes to the running cluster. Clients will update themselves automatically to connect to the new node once it has successfully joined the cluster.



Rebalancing of data (primary plus backup) on arrival or departure (forced or unforced) of a node is an automated process and no manual intervention is required.



You can promote your lite members to become data members. To do this, either use the Cluster API or the Management Center UI.



Setting hazelcast.initial.min.cluster.size to 4 & starting members one by one (empty cluster, no operation) can result in unexpected cluster partitioning behaviour: After reaching hazelcast.initial.min.cluster.size, partition table arrangement is initialized even though there's no data. When used in large clusters (>100) this adds an unnecessary overhead of partition assignment on each member addition.

**Further Reading:** * Online Documentation

1. Check that you have configured an adequate backup count based on your SLAs.
2. When using distributed computing features such as IExecutorService, EntryProcessors, Map/Reduce or Aggregators, any change in application logic or in the implementation of the above features must also be installed on member nodes. All the member nodes must be restarted after new code is deployed using the typical cluster re-deployment process:
   a. Shutdown servers
   b. Deploy the new application jar in the servers' classpath
   c. Start servers

# 27. CP Subsystem Overview

CP Subsystem is a component of a Hazelcast cluster that builds a strongly consistent layer for a set of distributed data structures. Its APIs can be used for implementing distributed coordination use

cases, such as leader election, distributed locking, synchronization, and metadata management. Its data structures are CP with respect to the CAP principle.

When running in CP Mode, only a select subset of members of a Hazelcast cluster take part in CP Subsystem. The number of Hazelcast members that take part in CP Subsystem is specified with the CP Subsystem Configuration parameter CPMemberCount. Knowing the number is important in order to know when you CP cluster is completely operational and also to determine when it loses or gains members. Each CP group elects its own Raft leader and runs the Raft consensus algorithm independently.

CP Subsystem runs a periodic background task to ensure that each CP member performs the Raft leadership role for roughly equal number of CP groups. If a CP member gets the Raft leadership role for too many CP groups compared to other CP members, it can turn into a bottleneck and these should be closely monitored and logged.

The CP Subsystem works only in memory without persisting any state to disk therefore a crashed CP member does not join to the cluster back by restoring its previous state. Therefore, crashed CP members poses s risk to the CP group and can lead to a gradual losing of majority of CP groups and eventually cause the total loss of availability of CP Subsystem. To mitigate these scenarios, crashed CP members should be removed from CP Subsystem and replaced in CP groups with other available CP members.



CP Subsystem operates in the unsafe mode by default. When CP Subsystem is not enabled, i.e CP Subsystem memberCount is 0, CP data structures operate in the unsafe mode without the strong consistency guarantee. To safely configure manage and recover from CP Sysbystem crash situations, please review the following section on CP Subsystem Persistence.

## 27.1. CP Subsystem Example Declarative Configuration:

```
<hazelcast>
    ...
    <cp-subsystem>
        <cp-member-count>7</cp-member-count>
        <group-size>3</group-size>
        <session-time-to-live-seconds>300</session-time-to-live-seconds>
        <session-heartbeat-interval-seconds>5</session-heartbeat-interval-seconds>
        <missing-cp-member-auto-removal-seconds>14400</missing-cp-member-auto-removal-
seconds>
        <fail-on-indeterminate-operation-state>false</fail-on-indeterminate-operation-
state>
        <persistence-enabled>true</persistence-enabled>
        <base-dir>/custom-cp-dir</base-dir>
    </cp-subsystem>
    ...
</hazelcast>
```

For more detailed information on CP Subsystem please review the following Reference.

**Further Reading:** * Online Documentation, Release Notes on CP Subsystem : CP Subsystem

## 27.2. CP Subsystem Persistence

CP Subsystem Persistence enables CP members to recover from crash scenarios. This capability significantly improves the overall reliability of CP Subsystem. This can be enabled by setting the CP Subsystem Configuration Parameter persistenceEnabled to true. This will allow CP members to persist their local state to stable storage and when the CP susbsytem is restarted after a crash, CP members can restore their local state from the persisted details and resume working as if they have never crashed.

**Further Reading:** * Online Documentation, Release Notes on CP Subsystem : * CP Subsystem Persistence

# 28. Rolling Member Upgrades

Rolling Member Upgrades feature in Hazelcast IMDG Enterprise allows the upgrading the version of Hazelcast members in a running cluster without interrupting the operation of the cluster. This involves selectively shutting down a node at a time, upgrading the node then restarting the node and repeating the same process until all nodes have been upgraded to the new version. This feature provides for the option of un-interrupted processing even when you require updates and upgrades of your cluster.

## 28.1. Rolling Upgrades Compatibility

There are some important caveats around Rolling Upgrades. When planning and executing rolling upgrades you should take care to understand Hazelcast versioning system as well as your current versions and your target version. Hazelcast members operating on binaries of the same major and minor version numbers are compatible regardless of patch version. E.g. in a cluster with members running on version 3.11.1, it is possible to perform a rolling upgrade to 3.11.2 by shutting down, upgrading to hazelcast-3.11.2.jar binary and starting each member one by one. Patch level compatibility applies to both Hazelcast IMDG and Hazelcast IMDG Enterprise.

Similarly, each minor version is compatible with the previous one (back until Hazelcast IMDG 3.8). For example, it is possible to perform a rolling upgrade on a cluster running Hazelcast IMDG Enterprise 3.11 to Hazelcast IMDG Enterprise 3.12. Rolling upgrades across minor versions is a Hazelcast IMDG Enterprise feature.

## 28.2. Rolling Upgrade Procedure

Suppose you are running a cluster with four members running on codebase version 3.12.0 with cluster version 3.12. You need upgrade to codebase version 3.13.0 and cluster version 3.13. The rolling upgrade process for this cluster, in general, would follows as described below to replace existing 3.12.0 members one by one with an upgraded one at version 3.13.0 includes the following steps.

- Gracefully shut down an existing 3.12.0 member.
- Wait until all partition migrations are completed; during migrations, membership changes (member joins or removals) are not allowed.
- Update the member with the new 3.13.0 Hazelcast binaries.

Start the member and wait until it joins the cluster. You should see something like the following in your logs: ... INFO: [192.168.2.2]:5701 [cluster] [3.13] Hazelcast 3.9 (20170630 - a67dc3a) starting at [192.168.2.2]:5701 ... INFO: [192.168.2.2]:5701 [cluster] [3.13] Cluster version set to 3.12

- The version in brackets ([3.13]) still denotes the member's codebase version (running on the hypothetical hazelcast-3.13.jar binary).
- Once the member locates the existing cluster members, it sends its join request to the master.
- The master validates that the new member is allowed to join the cluster and lets the new member know that the cluster is currently operating at 3.12 cluster version.
- The new member sets 3.12 as its cluster version and starts operating normally.

At this point all members of the cluster have been upgraded to codebase version 3.13.0 but the cluster still operates at cluster version 3.12. In order to use 3.13 features the cluster version must be changed to 3.13.

These steps should be repeated for each member node until all nodes are successfully upgraded.

For more detailed information on Logging features please review the following Reference.

**Further Reading:** * Online Documentation, Release Notes on Logging features : Rolling Upgrades

## 28.3. Upgrading Cluster Version

You can also upgrade the cluster version in IMDG. You can upgrade cluster versions by :

- Using Management Center.
- Using the cluster.sh script. in the bin folder
- Allow the cluster to auto-upgrade.

Note that you need to enable the REST API to use either of the above methods to upgrade your cluster version. The cluster write REST endpoint is disabled by default. To enable the CLUSTER_WRITE REST endpoint group.

Note that you need to upgrade your Management Center version before upgrading the member version if you want to change cluster version using Management Center. Management Center is compatible with the previous minor version of Hazelcast.

## 28.4. Enabling Auto-Upgrading

A cluster can automatically upgrade its version. When a cluster detects that all its members have a version higher than the current cluster version, it upgrades. The cluster version will then match the

member version. This feature is disabled by default. To enable it, set the system property hazelcast.cluster.version.auto.upgrade.enabled to true.

NOTE : There is one important detail to note. When shutting down and upgrading the members one by one, when you shut down the last one, all the members in the remaining cluster have the newer version. However if you might not want the auto-upgrade to kick in before you have successfully upgraded the last member as well. To avoid this scenario, you can use the hazelcast.cluster.version.auto.upgrade.min.cluster.size system property. You should set it to the size of your cluster, and then Hazelcast will wait until the last member is upgraded and joins the cluster and the correct size is noted before it can proceed with the auto-upgrade.

## 28.5. Network Partitions and Rolling Upgrades

In the event that your cluster is split your cluster into two sub-clusters resulting in a split brain while in the process of rolling upgrades some uneven version mismatch would be noted on the cluster. It is also important to note that two sub-clusters with differing version, will only merge as long as they operate on the same cluster version. This is a requirement to ensure that all members participating in each one of the sub-clusters are able to operate as members of the merged cluster at the same cluster version.

In order for the two subcultures to merge, it is necessary to change the cluster version of the sub-cluster that still operates on the old cluster version, so that both sub-clusters will be operating at the same, upgraded cluster version and able to merge as soon as the network partition is fixed.

# 29. Failure Detection and Recovery

The failure detector mechanism is responsible to determine if a member in the cluster is unreachable or crashed. The crucial detail in failure detection is to distinguish whether a member is still alive but slow or has crashed. By the FLP result theory, it is impossible to distinguish a crashed member from a slow one in an asynchronous system. A workaround to this limitation is to use unreliable failure detectors. An unreliable failure detector allows a member to suspect that others have failed, usually based on liveness criteria but it can make mistakes to a certain degree. This is the strategy employed by Hazelcast IMDG

Hazelcast has the following built-in failure detectors:

- Deadline Failure Detector
- Phi Accrual Failure Detector.
- Ping Failure Detector, that, if enabled, works in parallel with the above ones (This detector is by disabled default).

Note: Hazelcast also offers failure detectors for its Java client as well.

Generally, smooth and predictable operations are the norm, the occasional failure of hardware and software is inevitable. With the right detection, alerts, and recovery processes in place, your cluster will tolerate failure without incurring unscheduled downtime.

## 29.1. Common Causes Of Node Failure

The most common causes of node failure are garbage collection pauses and network connectivity issues. Both of these can cause a node to fail to respond to health checks and thus be removed from the cluster.

## 29.2. Failure Detection on Cluster Members

A failure detector is responsible for determining if a member of the cluster is unreachable or has crashed. Hazelcast IMDG has three built-in failure detectors; Deadline Failure Detector, Phi Accrual Failure Detector, and Ping Failure Detector.

### 29.2.1. Deadline Failure Detector

Deadline Failure Detector uses an absolute timeout for missing/lost heartbeats. After a timeout, a member is considered as crashed/unavailable and marked as suspected. Deadline Failure Detector is the default failure detector in Hazelcast IMDG.

This detector is also available in all Hazelcast client implementations. The deadline failure detector uses ICMP by default. However when this not possible it always falls back on TCP Echo attempts on port 7.

### 29.2.2. Phi Accrual Failure Detector

Phi Accrual Failure Detector is the failure detector based on The Phi Accrual Failure Detector' by Hayashibara et al. (https://www.computer.org/csdl/proceedings/srds/2004/2239/00/22390066-abs.html) It keeps track of the intervals between heartbeats in a sliding window of time and measures the mean and variance of these samples and calculates a value of suspicion level (Phi). The value of phi will increase when the period since the last heartbeat gets longer. If the network becomes slow or unreliable, the resulting mean and variance will increase, and thus there will need to be a longer period for which no heartbeat is received before the member is suspected.

### 29.2.3. Ping Failure Detector

The Ping Failure Detector was introduced in 3.9.1. It may be configured in addition to the Deadline or Phi Accrual Failure Detectors. It operates at Layer 3 of the OSI protocol and provides fast deterministic detection of hardware and other lower-level events. This detector may be configured to perform an extra check after a member is suspected by one of the other detectors, or it can work in parallel, which is the default. This way hardware and network-level issues will be detected more quickly.

This failure detector is based on `InetAddress.isReachable()`. When the JVM process has enough permissions to create RAW sockets, the implementation will choose to rely on ICMP Echo requests. This is preferred.

This detector is by default disabled and also available for Hazelcast Java Client.

**Further Reading:**

- Online Documentation, Failure Detector Configuration

## 29.3. Health Monitoring & Alerts

Please review Hazelcast Health Monitor logs regularly to understand you system runtime stability. Monitor Hazelcast logs and scan for failure messages regularly. Any incidences should be investigated and documented so as to understand your network better and find ways to provision, configure and manage your network more efficiently

Hazelcast IMDG provides multi-level tolerance configurations in a cluster:

1. Garbage collection tolerance—When a node fails to respond to health check probes on the existing socket connection but is actually responding to health probes sent on a new socket, it can be presumed to be stuck either in a long GC or in another long-running task. Adequate tolerance levels configured here may allow the node to come back from its stuck state within permissible SLAs.
2. Network tolerance—When a node is temporarily unreachable by any means, temporary network communication errors may cause nodes to become unresponsive. In such a scenario, adequate tolerance levels configured here will allow the node to return to healthy operation within permissible SLAs.

See System Properties for more details.

You should establish tolerance levels for garbage collection and network connectivity and then set monitors to raise alerts when those tolerance thresholds are crossed. Customers with a Hazelcast IMDG subscription can use the extensive monitoring capabilities of the Management Center to set monitors and alerts.

In addition to the Management Center, we recommend that you use jstat and keep verbose GC logging turned on and use a log scraping tool like Splunk or similar to monitor GC behavior. Back-to-back full GCs and anything above 90% heap occupancy after a full GC should be cause for alarm.

Hazelcast IMDG dumps a set of information into the console of each instance that may further be used to create alerts. The following is a detail of those properties:

- `processors`: The number of available processors in the machine
- `physical.memory.total`: Total memory
- `physical.memory.free`: Free memory
- `swap.space.total`: Total swap space
- `swap.space.free`: Available swap space
- `heap.memory.used`: Used heap space
- `heap.memory.free`: Available heap space
- `heap.memory.total`: Total heap memory
- `heap.memory.max`: Max heap memory
- `heap.memory.used/total`: The ratio of used heap to total heap

- `heap.memory.used/max`: The ratio of used heap to max heap
- `minor.gc.count`: The number of minor GCs that have occurred in JVM
- `minor.gc.time`: The duration of minor GC cycles
- `major.gc.count`: The number of major GCs that have occurred in JVM
- `major.gc.time`: The duration of all major GC cycles
- `load.process`: The recent CPU usage for the particular JVM process; a negative value if not available
- `load.system`: The recent CPU usage for the whole system; a negative value if not available
- `load.systemAverage`: The system load average for the last minute. The system load average is the sum of the number of runnable entities queued to the available processors and the number of entities running on available processors averaged over a period of time
- `thread.count`: The number of threads currently allocated in the JVM
- `thread.peakCount`: The peak number of threads allocated in the JVM
- `event.q.size`: The size of the event queue



Hazelcast IMDG uses internal executors to perform various operations that read tasks from a dedicated queue. Some of the properties below belong to such executors:

- `executor.q.async.size`: Async Executor Queue size. Async Executor is used for async APIs to run user callbacks and is also used for some Map/Reduce operations.
- `executor.q.client.size`: Size of Client Executor: Queue that feeds to the executor that perform client operations
- `executor.q.query.size`: Query Executor Queue size: Queue that feeds to the executor that execute queries
- `executor.q.scheduled.size`: Scheduled Executor Queue size: Queue that feeds to the executor that performs scheduled tasks
- `executor.q.io.size`: IO Executor Queue size: Queue that feeds to the executor that performs I/O tasks
- `executor.q.system.size`: System Executor Queue size: Executor that processes system-like tasks for cluster/partition
- `executor.q.operation.size`: The number of pending operations. When an operation is invoked, the invocation is sent to the correct machine and put in a queue to be processed. This number represents the number of operations in that queue
- `executor.q.priorityOperation.size`: Same as executor.q.operation.size. There are two types of operations – normal and priority. Priority operations end up in a separate queue
- `executor.q.response.size`: The number of pending responses in the response queue. Responses from remote executions are added to the response queue to be sent back to the node invoking the operation (e.g., the node sending a map.put for a key it does not own)
- `operations.remote.size`: The number of invocations that need a response from a remote

Hazelcast IMDG server instance

- `operations.running.size`: The number of operations currently running on this node
- `proxy.count`: The number of proxies
- `clientEndpoint.count`: The number of client endpoints
- `connection.active.count`: The number of currently active connections
- `client.connection.count: The number of current client connections

## 29.4. Recovery from a Partial or Total Failure

Under normal circumstances, Hazelcast IMDG members are self-recoverable as in the following scenarios:

- Automatic split-brain resolution
- Hazelcast IMDG allowing stuck/unreachable nodes to come back within configured tolerance levels (see above in the document for more details)

However, in the rare case when a node is declared unreachable by Hazelcast IMDG because it fails to respond, but the rest of the cluster is still running, use the following procedure for recovery:

1. Collect Hazelcast IMDG server logs from all server nodes, active and unresponsive.
2. Collect Hazelcast IMDG client logs or application logs from all clients.
3. If the cluster is running and one or more member nodes were ejected from the cluster because it was stuck, take a heap dump of any stuck member nodes.
4. If the cluster is running and one or more member nodes were ejected from the cluster because it was stuck, take thread dumps of server nodes including any stuck member nodes. For taking thread dumps, you may use the Java utilities `jstack`, `jconsole` or any other JMX client.
5. If the cluster is running and one or more member nodes were ejected from the cluster because it was stuck, collect `nmon` logs from all nodes in the cluster.
6. After collecting all of the necessary artifacts, shut down the rogue node(s) by calling shutdown hooks (see next section, Cluster Member Shutdown, for more details) or through JMX beans if using a JMX client.
7. After shutdown, start the server node(s) and wait for them to join the cluster. After successfully joining, Hazelcast IMDG will rebalance the data across the new nodes.



Hazelcast IMDG allows persistence based on Hazelcast IMDG callback APIs, which allow you to store cached data in an underlying data store in a write-through or write-behind pattern and reload into the cache for cache warm-up or disaster recovery. See this link for more details: Hot Restart Persistence

**Cluster Member Shutdown**

- `HazelcastInstance.shutdown()` is graceful so it waits for all backups to be completed. You may also use the web-based user interface in the Management Center to shut down a particular cluster member. See the Management Center section above for details.

- Make sure to shut down the Hazelcast IMDG instance on shutdown; in a web application, do it in a context destroy event - http://blog.hazelcast.com/pro-tip-shutdown-hazelcast-on-context-destroy/
- To perform a graceful shutdown in a web container, see http://stackoverflow.com/questions/18701821/hazelcast-prevents-the-jvm-from-terminating – Tomcat hooks; a Tomcat-independent way to detect JVM shutdown and safely call Hazelcast.shutdownAll().
- If an instance crashes or you are forced to shut it down ungracefully, any data that is unwritten to the cache, any enqueued write-behind data, and any data that has not yet been backed up will be lost.

## 29.5. Recovery from Client Connection Failures

When a client is disconnected from the cluster, it automatically tries to re-connect. There are configurations you can perform to achieve proper behavior. Please refer to the "Lazy Initiation and Connection Strategies" section of this document for further details about those behaviors.

While the client is trying to connect initially to one of the members in the cluster, none the members might be not available. In this case, you can configure the client to act in several ways:

- Client can give up, throwing an exception and shutting down eventually.
- Client will not shutdown, but will not block the operations and throw `HazelcastClientOfflineException` until it can reconnect.
- Client will block operations and retry as many as a fixed `connectionAttemptLimit` times, or retry with an exponential backoff mechanism based on users configuration.

**Further Reading:**

- Online Documentation: Setting Connection Attempt Limit
- Online Documentation: Configuring Client Connection Retry
- Online Documentation: Java Client Connection Strategy

# 30. Logging Overview

Hazelcast IMDG logging configuration and does not depend on any logging framework except JDK logging. It has built-in adapters for a number of logging frameworks and it also supports custom loggers by providing logging interfaces.

Log files are fundamental in monitoring, diagnosis and understanding of the health of a cluster. Log file should be well configured, easily discoverable, routinely backed up and readily provided as a basic resource for investigating issues.

Outside of PROD environments, extensive and debug logging is recommended.

## 30.1. Managing your Logs

When problems arise in a cluster the main objective is to understand the root cause of a problem and find a satisfactory resolution as quickly as possible. In most cases the quantity and quality of details in the logs provided are directly proportional to the time it takes to correctly evaluate the issue, categorize the issue and assign the appropriate expertise needed to analyze and help resolve the problem.

Hazelcast generates the following log files which should be managed on your system and made available to diagnose problems

- Hazelcast IMDG Log file
- Hazelcast Diagnostics Log file. Diagnostic log file depends on diagnostic logging enabled.
- Hazelcast Health Monitor logs.

In addition you can also configure Java on Hazelcast IMDG Instances to collect:

- Java Garbage collection log files.
- Heap Dumps
- Thread Dumps.

### 30.1.1. Monitoring and Using your Logs

Always monitor you log files, you can filter them to check for errors or warnings. Review the health monitor logs to ascertain the health of the system. Review diagnostic logs to understand how the system is managing and using resources and objects, and Review IMDG logs for overall process flow.

### 30.1.2. Dynamic Log Level Support

Starting with version 4.2, Hazelcast IMDG supports changing log levels without the need of restarting the cluster members. This feature is only supported for the default (JDK/JUL), Log4j, and Log4j2 frameworks. Either of the following ways make it possible to dynamically change the level of your cluster's logs;

- Using JMX API: The logging service exposes its JMX MBean as LoggingServiceMBean. You can retrieve, set, and reset the level. See the JMX API and Monitoring with JMX sections.
- Using REST API: You can use the /hazelcast/rest/log-level REST endpoint to retrieve (GET), set (POST), and reset (DELETE) the level. See the REST Endpoint Groups section.

### 30.1.3. Client Log File

Client logs and application logs should be configured to write to their own separate logs file. This aids in making sure that IMDG message sand issues can be isolated in specific areas and can be easily analyzed without having to deduce the source of the message.

Extensive logging for client and applications is also recommended.

For more detailed information on Logging features please review the following Reference. **Further Reading:**

- Online Documentation, Release Notes on Logging features :

Hazelcast Logging

## 30.2. Hazelcast IMDG Diagnostics Log

Hazelcast IMDG has an extended set of diagnostic plugins for both client and server. The diagnostics log is a more powerful mechanism than the health monitor, and a dedicated log file is used to write the content. A rolling file approach is used to prevent taking up too much disk space.

### 30.2.1. Enabling Diagnostics

On the member side the following parameters need to be added:

```
-Dhazelcast.diagnostics.enabled=true
-Dhazelcast.diagnostics.metric.level=info
-Dhazelcast.diagnostics.invocation.sample.period.seconds=30
-Dhazelcast.diagnostics.pending.invocations.period.seconds=30
-Dhazelcast.diagnostics.slowoperations.period.seconds=30
-Dhazelcast.diagnostics.directory=/<path-to-log-files-location>/logs
```

On the client side the following parameters need to be added:

```
-Dhazelcast.diagnostics.enabled=true
-Dhazelcast.diagnostics.metric.level=info
```

You can use this parameter to specify the location for log file:

```
-Dhazelcast.diagnostics.directory=/path/to/your/log/directory
```

This can run in production without significant overhead. Currently there is no information available regarding data-structure (e.g. IMap or IQueue) specifics.

The diagnostics log files can be sent, together with the regular log files, to Hazelcast for analysis.

For more information about the configuration options, see the class `com.hazelcast.internal.diagnostics.Diagnostics` and the surrounding plugins.

### 30.2.2. Plugins

The Diagnostic system works based on plugins.

#### BuildInfo

The build info plugin shows the details about the build. It shows not only the Hazelcast IMDG version and if the Enterprise version is enabled, it also shows the git revision number. This is especially important if you use SNAPSHOT versions.

Every time a new file in the rolling file appender sequence is created, the BuildInfo will be printed in the header. The plugin has very low overhead and can't be disabled.

#### System Properties

The System properties plugin shows all properties beginning with:

- java (excluding java.awt)
- hazelcast
- sun
- os

Because filtering is applied, the content of the diagnostics log is at low risk of catching all kinds of private information. It will also include the arguments that have been used to start up the JVM, even though this is not officially a system property.

Every time a new file in the rolling file appender sequence is created, the system properties will be printed in the header. The system properties plugin is very useful for a lot of things, including getting information about the OS and JVM. The plugin has very low overhead and can't be disabled.

#### Config Properties

The Config Properties plugin shows all Hazelcast IMDG properties that have been explicitly set (either on the command line or in the configuration).

Every time a new file in the rolling file appender sequence is created, the Config Properties will be printed in the header. The plugin has very low overhead and can't be disabled.

#### Metrics

Metrics is one of the richest plugins because it provides insight into what is happening inside the Hazelcast IMDG system. The metrics plugin can be configured using the following properties:

- `hazelcast.diagnostics.metrics.period.seconds`: The frequency of dumping to file. Its default value is 60 seconds.
- `hazelcast.diagnostics.metrics.level`: The level of metric details. Available values are MANDATORY, INFO, and DEBUG. Its default value is MANDATORY.

#### Slow Operations

The Slow Operation plugin detects two things:

- slow operations. This is the actual time an operation takes. In technical terms, this is the service time.

- slow invocations. The total time an invocation takes, including all queuing, serialization and deserialization, and the execution of an operation.

The Slow Operation plugin shows all kinds of information about the type of operation and the invocation. If there is some kind of obstruction, e.g., a database call taking a lot of time and therefore the map get operation is slow, the get operation will be seen in the slow operations section. Any invocation that is obstructed by this slow operation will be listed in the slow invocations second.

This plugin can be configured using the following properties:

- `hazelcast.diagnostics.slowoperations.period.seconds`: Its default value is 60 seconds.
- `hazelcast.slow.operation.detector.enabled`: Its default value is true.
- `hazelcast.slow.operation.detector.threshold.millis`: Its default value is 1000 milliseconds.
- `hazelcast.slow.invocation.detector.threshold.millis`: Its default value is -1.

**Invocations**

The Invocations plugin shows all kinds of statistics about current and past invocations:

- The current pending invocations
- The history of invocations that have been invoked, sorted by sample count. Imagine a system is doing 90% Map gets and 10 % Map puts. For discussion's sake, we also assume a put takes as much time as a get and that 1,000 samples are made, then the PutOperation will show 100 samples and the GetOperation will show 900 samples. The history is useful for getting an idea of how the system is being used. Be careful, because the system doesn't distinguish between, e.g., one invocation taking ten minutes and ten invocations taking one minute. The number of samples will be the same.
- Slow history. Imagine EntryProcessors are used. These will take quite a lot of time to execute and will obstruct other operations. The Slow History collects all samples where the invocation took more than the 'slow threshold'. The Slow History will not only include the invocations where the operations took a lot of time, but it will also include any other invocation that has been obstructed.

The Invocations plugin will periodically sample all invocations in the invocation registry. It will give an impression of which operations are currently executing.

The plugin has very low overhead and can be used in production. It can be configured using the following properties:

- `hazelcast.diagnostics.invocation.sample.period.seconds`: The frequency of scanning all pending invocations. Its default value is 60 seconds.
- `hazelcast.diagnostics.invocation.slow.threshold.seconds`: The threshold when an invocation is considered to be slow. Its default value is 5 seconds.

#### Overloaded Connections

The overloaded connections plug-in is a debug plug-in, and it is dangerous to use in a production environment. It is used internally to figure out what is inside connections and their write queues when the system is behaving badly. Otherwise, the metrics plugin only exposes the number of items pending, but not the type of items pending. The overloaded connections plugin samples connections that have more than a certain number of pending packets, deserializes the content, and creates some statistics per connection.

It can be configured using the following properties:

- `hazelcast.diagnostics.overloaded.connections.period.seconds`: The frequency of scanning all connections. 0 indicates disabled. Its default value is 0.
- `hazelcast.diagnostics.overloaded.connections.threshold`: The minimum number of pending packets. Its default value is 10000.
- `hazelcast.diagnostics.overloaded.connections.samples`: The maximum number of samples to take. Its default value is 1000.

#### MemberInfo

The member info plugin periodically displays some aspects of the basic state of the Hazelcast IMDG member. For instance, it shows what the current members are, if it is the master, etc. It is useful to get a fast impression of the cluster without needing to analyze a lot of data.

The plugin has very low overhead and can be used in production. It can be configured using the following property:

- `hazelcast.diagnostics.memberinfo.period.seconds`: The frequency the member info is being printed. Its default value is 60.

#### System Log

The System Log plugin listens to what happens in the cluster and will display if a connection is added/removed, if a member is added/removed, or if there is a change in the lifecycle of the cluster. It is specially written to create some kind of sanity when a user is running into connection problems. It includes quite a lot of detail about why, e.g., a connection was closed. So if there are connection issues, please look at the System Log plugin before diving into the underworld called logging.

The plugin has very low overhead and can be used in production. Be aware that if the partitions are being logged you get a lot of logging noise.

- `hazelcast.diagnostics.systemlog.enabled`: Specifies if the plugin is enabled. Its default value is true.
- `hazelcast.diagnostics.systemlog.partitions`: Specifies if the plugin should display information about partition migration. Beware that if enabled, this can become pretty noisy, especially if there are many partitions. Its default value is false.

## 30.3. Enabling Garbage Collection Logs

Hazelcast IMDG has an extended set of diagnostic plugins for both client and server. The diagnostics log is a more powerful mechanism than the health monitor, and a dedicated log file is used to write the content. A rolling file approach is used to prevent taking up too much disk space.

### 30.3.1. Garbage Collection Logs

On the member side the following parameters need to be added:

```
jstat -gcutil -t <java-pid> 1000 1
```

You can also enable GC logging settings as shown in your startup scripts:

```
-verbose:gc
-Xloggc:gc.log
-XX:NumberOfGCLogFiles=10
-XX:GCLogFileSize=10M
-XX:+UseGCLogFileRotation
-XX:+PrintGCDetails
-XX:+PrintGCDateStamps
-XX:+PrintTenuringDistribution
-XX:+PrintGCApplicationConcurrentTime
-XX:+PrintGCApplicationStoppedTime
```

Also, instead of default Java 8 ParallelGC, it is recommended to switch to CMS GC, since it ensures a good tradeoff between the operations throughput and GC pauses:

```
-XX:CMSInitiatingOccupancyFraction=65
-XX:+UseParNewGC
-XX:+UseConcMarkSweepGC
```

You can use this parameter to specify the location for log file:

```
-Xloggc:/path/to/your/log/directory/gc.log
```

### 30.3.2. Java Heap Dumps

You can configure and collect heap dumps by configuring you application to generate heap dumps on error as shown.

```
if [ "x$MIN_HEAP_SIZE" != "x" ]; then
JAVA_OPTS="$JAVA_OPTS -Xms${MIN_HEAP_SIZE}"
fi

if [ "x$MAX_HEAP_SIZE" != "x" ]; then
JAVA_OPTS="$JAVA_OPTS -Xmx${MAX_HEAP_SIZE}"
fi

JAVA_OPTS="$JAVA_OPTS -XX:+UseG1GC -XX:+UseCompressedOops -XX:+UseGCLogFileRotation
-XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=20M -XX:+PrintGCDetails
-XX:+PrintGCDateStamps -Xloggc:/path/to/your/log/directory/hazelcast-gc.log.`date +%Y-
%m-%d-%H-%M` -XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=/path/to/your/log/directory/ -verbose:gc
-Dlog4j.configuration=file:/path/to/your/log/directory/log4j.properties
-Djava.security.egd=file:/dev/./urandom
-Djava.io.tmpdir=/path/to/your/tmp/directory/tmp/"
```

### 30.3.3. Java Thread Dumps

Generate a Java thread dump You can use the attached script to generate a thread dump or you can use a tool e.g. visualVm or JProfiler. There are the steps for generating a Java thread dump:

1. Note the process ID number of the Java process (e.g. using top, a grep on ps -axw, etc.)
2. Send a QUIT signal to the process with the kill -QUIT or kill -3 command.

For example: Raw

```
kill -3 JAVA_PID
```

1. Run a verion of the sripgt below that is updated and ready for your environment
2. Make the script executable with chmod 755 NOTE It will capture a series of 6 thread dumps spaced 20 seconds apart (modify as needed), passing in the Java process ID as an argument. Please run generate the thread dumps when you see the close_wait sockets issue and before you downgrade the server.

```
#!/bin/sh
#
# Takes the Target Jav Process PID as an argument.
#
# Create thread dumps a specified number of times and INTERVAL. Thread dumps
# will be in the file where stdout was redirected or in console output.
#
# Usage: sh ./threaddump_linux.sh
#

# Number of times to collect data.
LOOP=6
# Interval in seconds between data points.
INTERVAL=20

for ((i=1; i <= $LOOP; i++))
do
   kill -3  $1
   echo "thread dump #" $i
   if [ $i -lt $LOOP ]; then
      echo "Sleeping..."
      sleep $INTERVAL
   fi
done
```

To run the script:

```
./threaddump.sh JAVA_PID
```

NOTE : Be sure to test the script before the issue happens to make sure it runs properly in your environment.

You can also use the JDK to generate Thread Dumps: . ThreadDump via JDK Use jps -lv to find the Java process ID for issuing kill -QUIT or kill -3. Be sure the -Xrs JVM option is not being used, as it causes SIGQUIT and SIGWAITING signals to be ignored. Running kill -3 sends a SIGQUIT signal to the JVM, so using this option will cause kill -3 to be ignored. If using OpenJDK or Sun JDK 1.6 or later, using jstack is an option. Execute the following, passing in the Java process ID: Raw

```
jstack -l JAVA_PID > jstack.out
```

- Most profilers have a feature to generate and browse thread dumps in runtime. For example:
  1. JConsole - http://openjdk.java.net/tools/svc/jconsole/
  2. VisualVM - https://visualvm.github.io/
  3. GlowRoot - https://github.com/glowroot/glowroot
  4. YourKit - http://www.yourkit.com/

5. JProfiler - http://www.ej-technologies.com/products/jprofiler/overview.html

# 31. Performance Tuning & Optimization

Aside from standard code optimization in your application, there are a few Hazelcast IMDG-specific optimizations to keep in mind when preparing for a new Hazelcast IMDG deployment.

## 31.1. Dedicated, Homogeneous Hardware Resources

The first, easiest, and most effective optimization strategy for Hazelcast IMDG is to ensure that Hazelcast IMDG services are allocated their own dedicated machine resources. Using dedicated, properly sized hardware (or virtual hardware) ensures that Hazelcast IMDG nodes have ample CPU, memory, and network resources without competing with other processes or services.

Hazelcast IMDG distributes load evenly across all of its member nodes and assumes that the resources available to each of its nodes are homogeneous. In a cluster with a mix of more and less powerful machines, the weaker nodes will cause bottlenecks, leaving the stronger nodes underutilized. For predictable performance, it is best to use equivalent hardware for all Hazelcast IMDG nodes.

## 31.2. Partition Count

Hazelcast IMDG's default partition count is 271. This is a good choice for clusters of up to 50 nodes and ~25–30 GB of data. Up to this threshold, partitions are small enough that any rebalancing of the partition map when nodes join or leave the cluster doesn't disturb the smooth operation of the cluster. With larger clusters and/or bigger data sets, a larger partition count helps to maintain an efficient rebalancing of data across nodes.

An optimum partition size is between 50MB – 100MB. Therefore, when designing the cluster, determine the size of the data that will be distributed across all nodes, and then determine the number of partitions such that no partition size exceeds 100MB. If the default count of 271 results in heavily loaded partitions, increase the partition count to the point where data load per-partition is under 100MB. Remember to factor in headroom for projected data growth.



If you change the partition count from the default of 271, be sure to use a prime number of partitions. This will help minimize collision of keys across partitions, ensuring more consistent lookup times. For further reading on the advantages of using a prime number of partitions, see http://www.quora.com/Does-making-array-size-a-prime-number-help-in-hash-table-implementation-Why.



Important: If you are an Enterprise customer using the High-Density Data Store with large data sizes, we recommend a large increase in partition count, starting with 5009 or higher.

The partition count cannot be changed after a cluster is created, so if you have a larger cluster, be sure to test and set an optimum partition count prior to deployment. If you need to change the

partition count after a cluster is running, you will need to schedule a maintenance window to update the partition count and restart the cluster.

## 31.3. Parallel Migration

Starting with version 4.1, partition migrations for separate partitions can be done in parallel. This significantly reduces the time needed for repartitioning. Having parallel migrations does have an impact on heap memory and network utilization, so we have introduced the following properties to control the number of parallel migrations/replications on a single member:

- `hazelcast.partition.max.parallel.migrations` - Maximum number of partition migrations to be executed concurrently on a member
- `hazelcast.partition.max.parallel.replications` - Maximum number of parallel partition backup replication operations per member

The default value is 10 for both properties, and this should be fine for most setups. Care and consideration should be taken before altering either of these properties. Decreasing them will make the total migration time go up, and increasing them will create additional heap and network pressure.

## 31.4. Dedicated Network Interface Controller for Hazelcast IMDG Members

Provisioning a dedicated physical network interface controller (NIC) for Hazelcast IMDG member nodes ensures smooth flow of data, including business data and cluster health checks, across servers. Sharing network interfaces between a Hazelcast IMDG instance and another application could result in choking the port, thus causing unpredictable cluster behavior.

## 31.5. Network Settings

### 31.5.1. Adjust TCP buffer size

TCP uses a congestion window to determine how many packets it can send at one time; the larger the congestion window, the higher the throughput. The maximum congestion window is related to the amount of buffer space that the kernel allocates for each socket. For each socket, there is a default value for the buffer size, which may be changed by using a system library call just before opening the socket. The buffer size for both the receiving and sending sides of the socket may be adjusted.

To achieve maximum throughput, it is critical to use optimal TCP socket buffer sizes for the links you are using to transmit data. If the buffers are too small, the TCP congestion window will never open up fully, therefore throttling the sender. If the buffers are too large, the sender can overrun the receiver such that the sending host is faster than the receiving host, which will cause the receiver to drop packets and the TCP congestion window to shut down.

Hazelcast IMDG, by default, configures I/O buffers to 128KB, but these are configurable properties and may be changed in Hazelcast IMDG's configuration with the following parameters:

- hazelcast.socket.receive.buffer.size
- hazelcast.socket.send.buffer.size

Typically, throughput may be determined by the following formulae:

- TPS = Buffer Size / Latency
- Buffer Size = RTT (Round Trip Time) * Network Bandwidth

To increase TCP Max Buffer Size in Linux, see the following settings:

- net.core.rmem.max
- net.core.wmem.max

To increase TCP auto-tuning by Linux, see the following settings:

- net.ipv4.tcp.rmem
- net.ipv4.tcp.wmem

**Further Reading:**

- http://www.linux-admins.net/2010/09/linux-tcp-tuning.html

# 31.6. Garbage Collection

Keeping track of garbage collection statistics is vital to optimum Java performance, especially if you run the JVM with large heap sizes. Tuning the garbage collector for your use case is often a critical performance practice prior to deployment. Likewise, knowing what baseline garbage collection behavior looks like and monitoring for behavior outside of normal tolerances will keep you aware of potential memory leaks and other pathological memory usage.

## 31.6.1. Minimize Heap Usage

The best way to minimize the performance impact of garbage collection is to keep heap usage small. Maintaining a small heap can save countless hours of garbage collection tuning and will provide improved stability and predictability across your entire application. Even if your application uses very large amounts of data, you can still keep your heap small by using Hazelcast IMDG's High-Density Memory Store. Some common off-the-shelf GC tuning parameters for Hotspot and OpenJDK:

```
-XX:+UseParallelOldGC
-XX:+UseParallelGC
-XX:+UseCompressedOops
```

To enable GC logging, use the following JVM arguments for Java 8

```
-verbose:gc
-Xloggc:gc.log
-XX:NumberOfGCLogFiles=10
-XX:GCLogFileSize=10M
-XX:+UseGCLogFileRotation
-XX:+PrintGCDetails
-XX:+PrintGCDateStamps
-XX:+PrintTenuringDistribution
-XX:+PrintGCApplicationConcurrentTime
-XX:+PrintGCApplicationStoppedTime
```

The above arguments to enable logging, only work for Java 8. For Java 9+ the following arguments can be used:

```
-Xlog:safepoint,gc+age=debug,gc*=debug:file=gc.log:uptime,level,tags:filesize=10m,file
count=10
```

## 31.7. High-Density Memory Store (Enterprise HD Feature)

Hazelcast IMDG's High-Density Memory Store (HDMS) is an in-memory storage option that uses native, off-heap memory to store object data instead of the JVM heap. This allows you to keep terabytes of data in memory without incurring the overhead of garbage collection. HDMS capabilities support JCache, Map, Hibernate, and Web Sessions data structures.

Available to Hazelcast IMDG Enterprise customers, the HDMS is an ideal solution for those who want the performance of in-memory data, need the predictability of well-behaved Java memory management, and don't want to spend time and effort on meticulous and fragile garbage collection tuning.



If you are an Enterprise HD customer using the HDMS with large data sizes, we recommend a large increase in partition count, starting with 5009 or higher. See the Partition Count section above for more information. Also, if you intend to pre-load very large amounts of data into memory (tens, hundreds, or thousands of gigabytes), be sure to profile the data load time and to take that startup time into account prior to deployment.

**Further Reading:**

- Online Documentation
- Hazelcast IMDG Resources

# 31.8. Azul Zing® and Zulu® Support (Enterprise Feature)

Azul Systems, the industry's only company exclusively focused on Java and the Java Virtual Machine (JVM), builds fully supported, certified standards-compliant Java runtime solutions that help enable real-time business. Zing is a JVM designed for enterprise Java applications and workloads that require any combination of low latency, high transaction rates, large working memory, and/or consistent response times. Zulu and Zulu Enterprise are Azul's certified, freely available open source builds of OpenJDK with a variety of flexible support options, available in configurations for the enterprise as well as custom and embedded systems. Starting with version 3.6, Azul Zing is certified and supported in Hazelcast IMDG Enterprise. When deployed with Zing, Hazelcast IMDG deployments gain performance, capacity, and operational efficiency within the same infrastructure. Additionally, you can directly use Hazelcast IMDG with Zulu without making any changes to your code.

**Further Information:**

- Webinar: https://hazelcast.com/resources/webinar-azul-systems-zing-jvm/

# 31.9. Pipelining

Starting with Hazelcast 3.12, the Pipelining feature can be used to send multiple requests in parallel on a single thread for increased throughput. This feature can be used with any asynchronous call and provides a cheap and disposable way to increase the overall throughput of a group of requests with built-in per-pipeline back pressure. More information on this feature can be found in the *Pipelining* section of the Hazelcast Reference Manual.

**Further Reading:**

- Online Documentation for Pipelining

# 31.10. Optimizing Queries

## 31.10.1. Add Indexes for Queried Fields

- For queries on fields with ranges, you can use an ordered index.

Hazelcast IMDG, by default, caches the deserialized form of the object under query in memory when inserted into an index. This removes the overhead of object deserialization per query, at the cost of increased heap usage.

## 31.10.2. Composite Indexes

Composite indexes are built on top of multiple map entry attributes; thus, increase the performance of complex queries significantly when used correctly.

**Further Reading**

- Online Documentation

### 31.10.3. Parallel Query Evaluation & Query Thread Pool

- Setting `hazelcast.query.predicate.parallel.evaluation` to `true` can speed up queries when using slow predicates or when there are > 100,000s entries per member.
- If you're using queries heavily, you can benefit from increasing query thread pools.

**Further Reading:**

- Online Documentation

### 31.10.4. OBJECT "in-memory-format"

Setting the queried entries' in-memory format to "OBJECT" will force that object to be always kept in object format, resulting in faster access for queries, but also in higher heap usage. It will also incur an object serialization step on every remote "get" operation.

**Further Reading:**

- Hazelcast Blog
- Online Documentation

### 31.10.5. Implement the "Portable" Interface on Queried Objects

The Portable interface allows for individual fields to be accessed without the overhead of deserialization or reflection and supports query and indexing support without full-object deserialization.

**Further Reading:**

- Hazelcast Blog
- Online Documentation

## 31.11. Optimizing Serialization

Hazelcast IMDG supports a range of object serialization mechanisms, each with their own costs and benefits. Choosing the best serialization scheme for your data and access patterns can greatly increase the performance of your cluster. An in-depth discussion of the various serialization methods is referenced below, but here is an at-a-glance summary:

**java.io.Serializable**

Benefits:

- Standard Java
- Does not require custom serialization implementation

Costs:

- Not as memory- or CPU-efficient as other options

**java.io.Externalizable**

Benefits over standard Java serialization:

- Allows client-provided implementation

Benefits:

- Standard Java
- More memory- and CPU-efficient than built-in Java serialization

Costs:

- Requires a custom serialization implementation

**com.hazelcast.nio.serialization.DataSerializable**

Optimization over standard Java Serialization:

- Doesn’t store class metadata

Benefits:

- More memory- and CPU-efficient than built-in Java serialization

Costs:

- Not standard Java
- Requires a custom serialization implementation
- Uses reflection

**com.hazelcast.nio.serialization.IdentifiedDataSerializable**

Optimization over standard Java Serialization

- Doesn’t use reflection

Benefits:

- Can help manage object schema changes by making object instantiation into the new schema from the older version’s instance explicit
- More memory-efficient than built-in Java serialization, more CPU-efficient than DataSerializable

Costs:

- Not standard Java
- Requires a custom serialization implementation
- Requires configuration and implementation of a factory method

**com.hazelcast.nio.serialization.Portable**

Optimization over other serialization schemes:

- Supports partial deserialization during queries

Benefits:

- More CPU-efficient than other serialization schemes in cases where you don't need access to the entire object
- Doesn't use reflection
- Supports versioning

Costs:

- Not standard Java
- Requires a custom serialization implementation
- Requires implementation of factory and class definition
- Class definition (metadata) is sent with object data, but only once per class

**Pluggable serialization libraries, e.g., Kryo**

Benefits:

- Convenient and flexible
- Can be stream or byte-array based

Costs:

- Often requires serialization implementation
- Requires plugin configuration. Sometimes requires class annotations

# 31.12. Serialization Optimization Recommendations

- Use `IMap.set()` on maps instead of `IMap.put()` if you don't need the old value. This eliminates unnecessary deserialization of the old value.
- Set "native byte order" and "allow unsafe" to "true" in the Hazelcast IMDG configuration. Setting the native byte array and unsafe options to true enables fast copy of primitive arrays like `byte[]`, `long[]`, etc., in your object.
- Compression—Compression is supported only by `Serializable` and `Externalizable`. It has not been applied to other serializable methods because it is much slower (around three orders of magnitude slower than not using compression) and consumes a lot of CPU. However, it can reduce binary object size by an order of magnitude.
- SharedObject—If set to "true", the Java serializer will back-reference an object pointing to a previously serialized instance. If set to "false", every instance is considered unique and copied separately even if they point to the same instance. The default configuration is false.

**Further Reading:**

- Tutorial
- Webinar
- Kryo Serializer
- Performance Top Five

# 31.13. Executor Service Optimizations

Hazelcast IMDG's IExecutorService is an extension of Java's built-in ExecutorService that allows for distributed execution and control of tasks. There are a number of options to Hazelcast IMDG's executor service that will have an impact on performance.

## 31.13.1. Number of Threads

An executor queue may be configured to have a specific number of threads dedicated to executing enqueued tasks. Set the number of threads appropriate to the number of cores available for execution. Too few threads will reduce parallelism, leaving cores idle, while too many threads will cause context switching overhead.

## 31.13.2. Bounded Execution Queue

An executor queue may be configured to have a maximum number of entries. Setting a bound on the number of enqueued tasks will put explicit back pressure on enqueuing clients by throwing an exception when the queue is full. This will avoid the overhead of enqueuing a task only for it to be canceled because its execution takes too long. It will also allow enqueuing clients to take corrective action rather than blindly filling up work queues with tasks faster than they can be executed.

## 31.13.3. Avoid Blocking Operations in Tasks

Any time spent blocking or waiting in a running task is thread execution time wasted while other tasks wait in the queue. Tasks should be written such that they perform no potentially blocking operations (e.g., network or disk I/O) in their `run()` or `call()` methods.

## 31.13.4. Locality of Reference

By default, tasks may be executed on any member node. Ideally, however, tasks should be executed on the same machine that contains the data the task requires to avoid the overhead of moving remote data to the local execution context. Hazelcast IMDG's executor service provides a number of mechanisms for optimizing locality of reference.

- Send tasks to a specific member—Using ExecutorService.executeOnMember(), you may direct execution of a task to a particular node
- Send tasks to a key owner—If you know a task needs to operate on a particular map key, you may direct execution of that task to the node that owns that key
- Send tasks to all or a subset of members—If, for example, you need to operate on all of the keys

in a map, you may send tasks to all members such that each task operates on the local subset of keys, then return the local result for further processing in a Map/Reduce-style algorithm

### 31.13.5. Scaling Executor Services

If you find that your work queues consistently reach their maximum and you have already optimized the number of threads and locality of reference and removed any unnecessary blocking operations in your tasks, you may first try to scale up the hardware of the overburdened members by adding cores and, if necessary, more memory.

When you have reached diminishing returns on scaling up (such that the cost of upgrading a machine outweighs the benefits of the upgrade), you can scale out by adding more nodes to your cluster. The distributed nature of Hazelcast IMDG is perfectly suited to scaling out, and you may find in many cases that it is as easy as just configuring and deploying additional virtual or physical hardware.

### 31.13.6. Executor Services Guarantees

In addition to the regular distributed executor service, durable and scheduled executor services have been added to the feature set of Hazelcast IMDG starting with versions 3.7 and 3.8. Note that when a node failure occurs, durable and scheduled executor services come with "at least once execution of a task" guarantee, while the regular distributed executor service has none.

## 31.14. Executor Service Tips and Best Practices

### 31.14.1. Work Queue Is Not Partitioned

Each member-specific executor will have its own private work-queue. Once a job is placed on that queue, it will not be taken by another member. This may lead to a condition where one member has a lot of unprocessed work while another is idle. This could be the result of an application call such as the following:

```
for(;;){
    iexecutorservice.submitToMember(mytask, member)
}
```

This could also be the result of an imbalance caused by the application, such as in the following scenario: all products by a particular manufacturer are kept in one partition. When a new, very popular product gets released by that manufacturer, the resulting load puts a huge pressure on that single partition while others remain idle.

### 31.14.2. Work Queue Has Unbounded Capacity by Default

This can lead to `OutOfMemoryError` because the number of queued tasks can grow without bounds. This can be solved by setting the `<queue-capacity>` property on the executor service. If a new task is submitted while the queue is full, the call will not block, but will immediately throw a `RejectedExecutionException` that the application must handle.

### 31.14.3. No Load Balancing

There is currently no load balancing available for tasks that can run on any member. If load balancing is needed, it may be done by creating an IExecutorService proxy that wraps the one returned by Hazelcast IMDG. Using the members from the `ClusterService` or member information from `SPI:MembershipAwareService`, it could route "free" tasks to a specific member based on load.

### 31.14.4. Destroying Executors

An IExecutorService must be shut down with care because it will shut down all corresponding executors in every member and subsequent calls to proxy will result in a `RejectedExecutionException`. When the executor is destroyed and later a `HazelcastInstance.getExecutorService` is done with the id of the destroyed executor, a new executor will be created as if the old one never existed.

### 31.14.5. Exceptions in Executors

When a task fails with an exception (or an error), this exception will not be logged by Hazelcast IMDG by default. This comports with the behavior of Java's ThreadPoolExecutorService, but it can make debugging difficult. There are, however, some easy remedies: either add a try/catch in your runnable and log the exception, or wrap the runnable/callable in a proxy that does the logging; the last option will keep your code a bit cleaner.

**Further Reading:**

- Mastering Hazelcast IMDG—Distributed Executor Service
- Online Documentation

## 31.15. Back Pressure

When using asynchronous calls or asynchronous backups, you may need to enable back pressure to prevent Out of Memory Exception (OOME).

**Further Reading:**

- Online Documentation

## 31.16. Entry Processors

Hazelcast IMDG allows you to update the whole or a part of IMap/ICache entries in an efficient and a lock-free way using Entry Processors.

- You can update entries for a given key/key set or filtered by a predicate. `Offloadable` and `ReadOnly` interfaces help to tune the Entry Processor for better performance.

**Further Reading:**

- Hazelcast IMDG Documentation, Entry Processor

- Mastering Hazelcast IMDG, Entry Processor
- Hazelcast IMDG Documentation, Entry Processor Performance Optimizations

## 31.17. Near Cache

Access to small-to-medium, read-mostly data sets may be sped up by creating a Near Cache. This cache maintains copies of distributed data in local memory for very fast access.

Benefits:

- Avoids the network and deserialization costs of retrieving frequently-used data remotely
- Eventually consistent
- Can persist keys on a filesystem and reload them on restart. This means you can have your Near Cache ready right after application start
- Can use deserialized objects as Near Cache keys to speed up lookups

Costs:

- Increased memory consumption in the local JVM
- High invalidation rates may outweigh the benefits of locality of reference
- Strong consistency is not maintained; you may read stale data

**Further Reading:**

- http://blog.hazelcast.com/pro-tip-near-cache/
- https://blog.hazelcast.com/fraud-detection-near-cache-example
- http://hazelcast.org/mastering-hazelcast/#near-cache
- http://docs.hazelcast.org/docs/latest/manual/html-single/index.html#near-cache
- http://docs.hazelcast.org/docs/latest/manual/html-single/index.html#configuring-near-cache
- http://docs.hazelcast.org/docs/latest/manual/html-single/index.html#jcache-near-cache
- http://docs.hazelcast.org/docs/latest/manual/html-single/index.html#configuring-client-near-cache

## 31.18. Client Executor Pool Size

The Hazelcast IMDG client uses an internal executor service (different from the distributed IExecutorService) to perform some of its internal operations. By default, the thread pool for that executor service is configured to be the number of cores on the client machine times five—e.g., on a 4-core client machine, the internal executor service will have 20 threads. In some cases, increasing that thread pool size may increase performance.

**Further Reading:**

- Online Documentation

## 31.19. Clusters with Many (Hundreds) of Nodes or Clients

Very large clusters of hundreds of nodes are possible with Hazelcast IMDG, but stability will depend heavily on your network infrastructure and ability to monitor and manage that many servers. Distributed executions in such an environment will be more sensitive to your application's handling of execution errors, timeouts, and the optimization of task code.

In general, you will get better results with smaller clusters of Hazelcast IMDG members running on more powerful hardware and a higher number of Hazelcast IMDG clients. When running large numbers of clients, network stability will still be a significant factor in overall stability. If you are running in Amazon's EC2, hosting clients and servers in the same zone is beneficial. Using Near Cache on read-mostly data sets will reduce server load and network overhead. You may also try increasing the number of threads in the client executor pool (see above).

**Further Reading:**

- Hazelcast Blog
- Hazelcast Blog
- Online Documentation

## 31.20. Linux Memory Management Recommendations

### 31.20.1. Disabling Transparent Huge Pages (THP)

Transparent Huge Pages (THP) is the Linux Memory Management feature which aims to improve the application performance by using the larger memory pages. In most of the cases it works fine but for databases and IMDGs it usually causes a significant performance drop. Since it's enabled on most of the Linux distributions, we do recommend disabling it when you run Hazelcast IMDG.

Use the following command to check if it's enabled:

```
cat /sys/kernel/mm/transparent_hugepage/enabled
cat /sys/kernel/mm/transparent_hugepage/defrag
```

Or an alternative command if you run RHEL:

```
cat /sys/kernel/mm/redhat_transparent_hugepage/enabled
cat /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

To disable it permanently, please see the corresponding docs for the Linux distribution that you use. Here is an example of the instructions for RHEL: https://access.redhat.com/solutions/46111

### 31.20.2. Disabling Swap Usage

Swapping behavior can be configured by setting the kernel parameter (/proc/sys/vm/swappiness) and can be turned off completely by executing 'swapoff -a' as root in Linux systems. We highly recommend turning off the swapping on the machines that run Hazelcast IMDG. When your operating systems start swapping, GC activities take much longer due to the low speed of disc access.

The Linux kernel parameter, `vm.swappiness`, is a value from 0-100 that controls the swapping of application data from physical memory to virtual memory on disk prevent Linux kernel to start swapping memory to disk way too early we need to set the default of 60 to value between 0 and 10. The higher the parameter value, the more aggressively inactive processes are swapped out from physical memory. The lower the value, the less they are swapped, forcing filesystem buffers to be emptied. In case swapping needs to be kept enabled, we recommend setting the value between 0 and 10 to prevent the Linux kernel to start swapping memory to disk way too early.

```
sudo sysctl vm.swappiness=10
```

## 31.21. Basic Optimization Recommendations

- 8 cores per Hazelcast IMDG server instance
- Minimum of 8 GB RAM per Hazelcast IMDG member (if not using the High-Density Memory Store)
- Dedicated NIC per Hazelcast IMDG member
- Linux—any distribution
- All member nodes should run within the same subnet
- All member nodes should be attached to the same network switch

## 31.22. Setting Internal Response Queue Idle Strategies

Starting with Hazelcast IMDG 3.7, there exists a special option that sets the response thread for internal operations on the client-server. By setting the backoff mode on and depending on the use case, you can get a 5-10% performance improvement. However, this will increase CPU utilization. To enable backoff mode please set the following property for Hazelcast IMDG cluster members:

```
-Dhazelcast.operation.responsequeue.idlestrategy=backoff
```

For Hazelcast IMDG clients, please use the following property to enable backoff:

```
-Dhazelcast.client.responsequeue.idlestrategy=backoff
```

## 31.23. TLS/SSL Performance Improvements for Java

TLS/SSL can have a significant impact on performance. There are a few ways to increase the performance. Please see the details for TLS/SSL performance improvements in the Online Documentation.

## 31.24. AWS Deployments

When you deploy Hazelcast IMDG clusters on AWS EC2 instances, you can consider to place the cluster members on the same Cluster Placement Group. This will help to reduce the latency among members drastically. Additionally, you can also consider using private IPs instead of public ones to increase the throughput when the cluster members are placed in the same VPC.

# 32. Management Center (Subscription and Enterprise Features)

The Hazelcast Management Center is a product available to Hazelcast IMDG Enterprise and Professional subscription customers that provides advanced monitoring and management of Hazelcast IMDG clusters. In addition to monitoring the overall cluster state, Management Center also allows you to analyze and browse your data structures in detail, update map configurations, and take thread dumps from nodes. With its scripting and console module, you can run scripts (JavaScript, Ruby, Groovy, and Python) and commands on your nodes.

## 32.1. Cluster-Wide Statistics and Monitoring

While each member node has a JMX management interface that exposes per-node monitoring capabilities, the Management Center collects all of the individual member node statistics to provide cluster-wide JMX and REST management APIs, making it a central hub for all of your cluster's management data. In a production environment, the Management Center is the best way to monitor the behavior of the entire cluster, both through its web-based user interface and through its cluster-wide JMX and REST APIs.

## 32.2. Web Interface Homepage

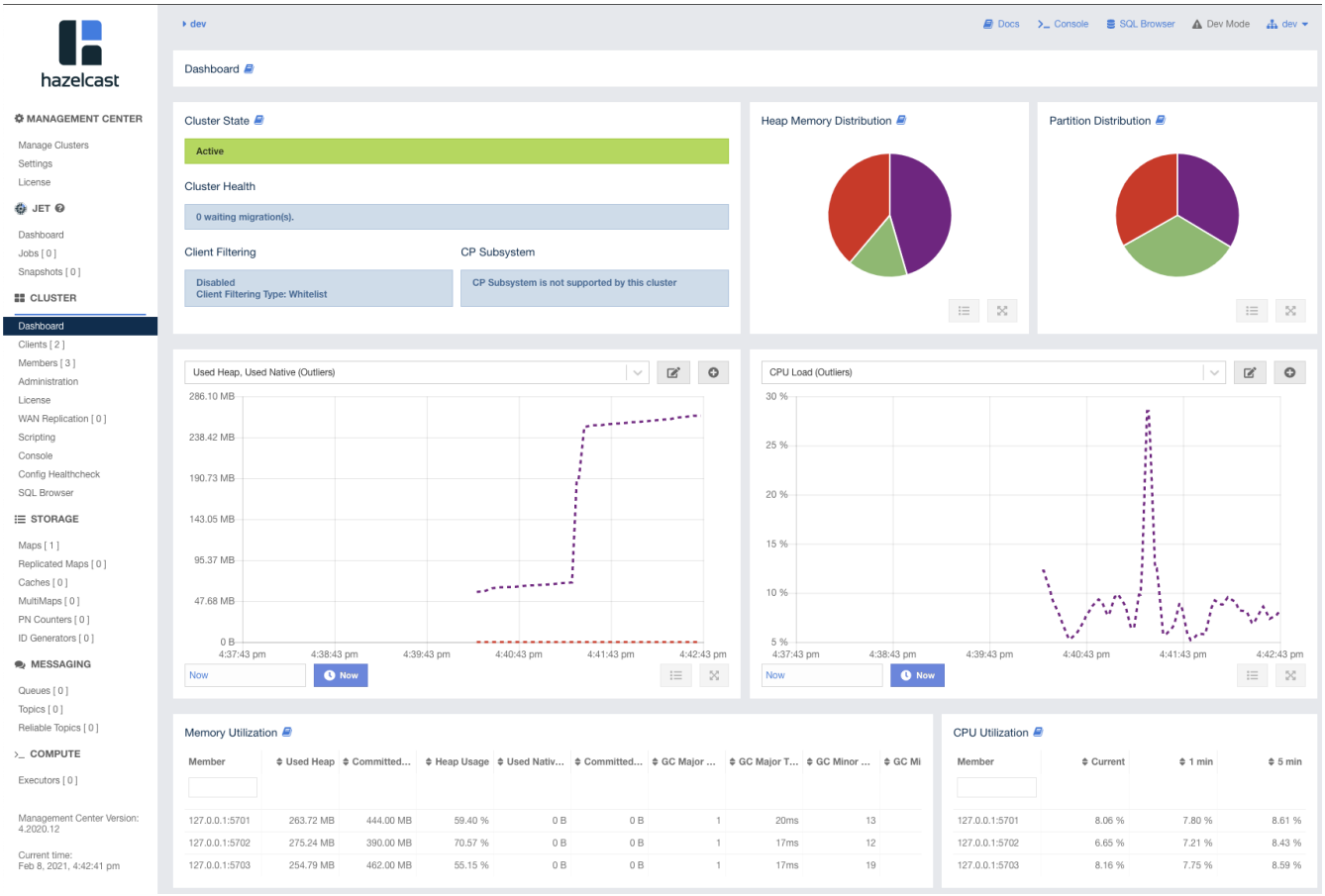



*Figure 4. Management Center Homepage*

The Dashboard is the first page appearing after logging in of the Management Center provides an overview of the connected cluster. For each node, it displays at-a-glance statistics that may be used to quickly gauge the status and health of each member and the cluster as a whole.

Dashboard Cluster State:

- Cluster State:
  - Active,
  - No Migration,
  - Frozen,
  - Passive,
  - In Transition
- Cluster Health: Shows how many migrations are taking place currently.
- Client Filtering:
  - Enabled/Disabled,
  - Client Filtering Type: WhileList, BlackList
- CP Subsystem:
  - CP Subsystem is not supported by this cluster,
  - CP Subsystem is not enabled, All CP members are accessible,
  - CP Subsystem warning: one CP member is not accessible,

- CP Subsystem alert: multiple CP members are not accessible,
- CP Subsystem error: minority of the CP members are not accessible,
- CP Subsystem error: majority of the CP members are not accessible.

Dashboard cluster-wide statistics:

- Heap Memory Distribution provides the cluster wise breakdown of heap memory.
- Partition Distribution char shows what percentage of partitions each cluster member has.

Memory Utilization per node:

- Used heap
- Committed heap
- Max heap
- Heap usage percentage
- Used native memory
- Committed native memory
- GC Major count
- GC Major time
- GC Minor count
- GC Minor time

CPU Utilization

The first column lists the members with their IPs and ports. The next columns list the system load averages on each member for the current moment, last 1 and 5 minutes.

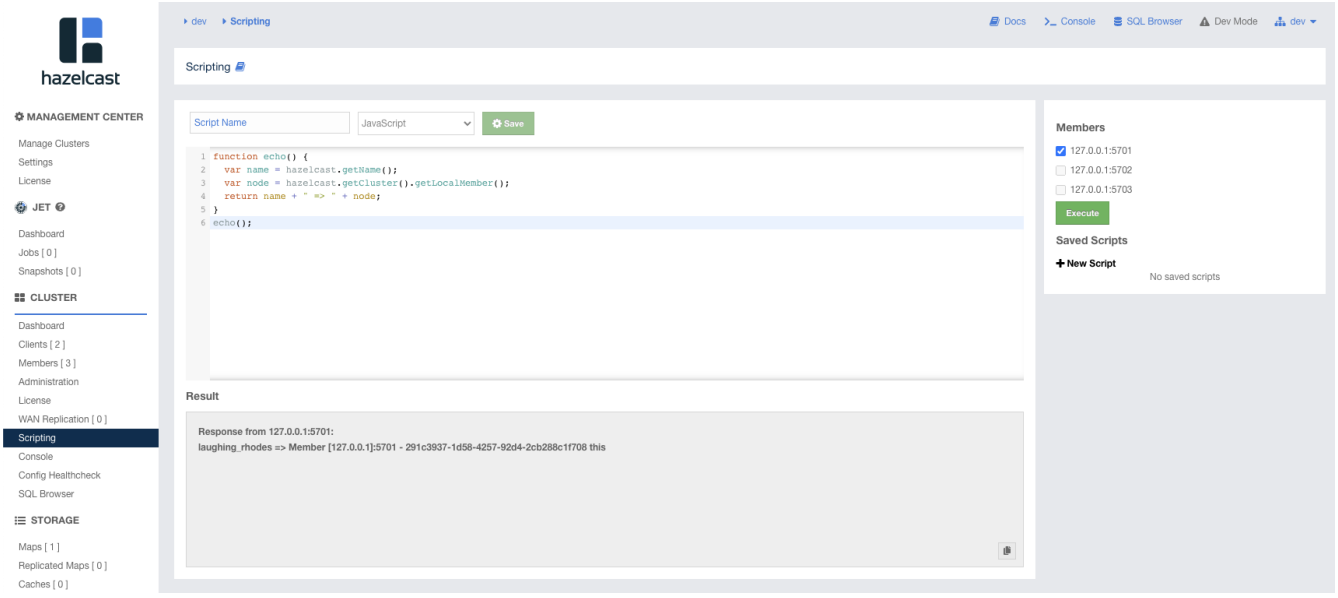



*Figure 5. Management Center Scripting*

The toolbar menu provides access to various resources and functions available in the Management Center. These include:

- MANAGEMENT CENTER: Includes menu items to manage clusters, user and license operations, and examine the user interface logs.
  - "Manage Clusters" is the first page that appears after you login to Management Center for the first time, you need to add your cluster's configuration using this page.
- JET: Includes menu items to export snapshots and manage Jet jobs.
- CLUSTER: Includes menu items for general cluster operations, monitoring clients/cluster members and executing scripts/commands.
  - Administration—provides user access management (available to admin users only)
  - Scripting—allows ad-hoc Javascript, Ruby, Groovy, or Python scripts to be executed against the cluster. Note that it's disabled by default and needs to enabled in cluster's configuration
  - Console—provides a terminal-style command interface to view information about and to manipulate cluster members and data structures
  - SQL Browser provides a panel that enabled to query IMDG cluster with SQL
- STORAGE: Includes menu items to monitor data structures such as the maps, caches and PN counters in your cluster.
- MESSAGING: Includes menu items to monitor queues and topics in your cluster.
- COMPUTE: Includes the "Executors" menu item to monitor the executor services in your cluster.

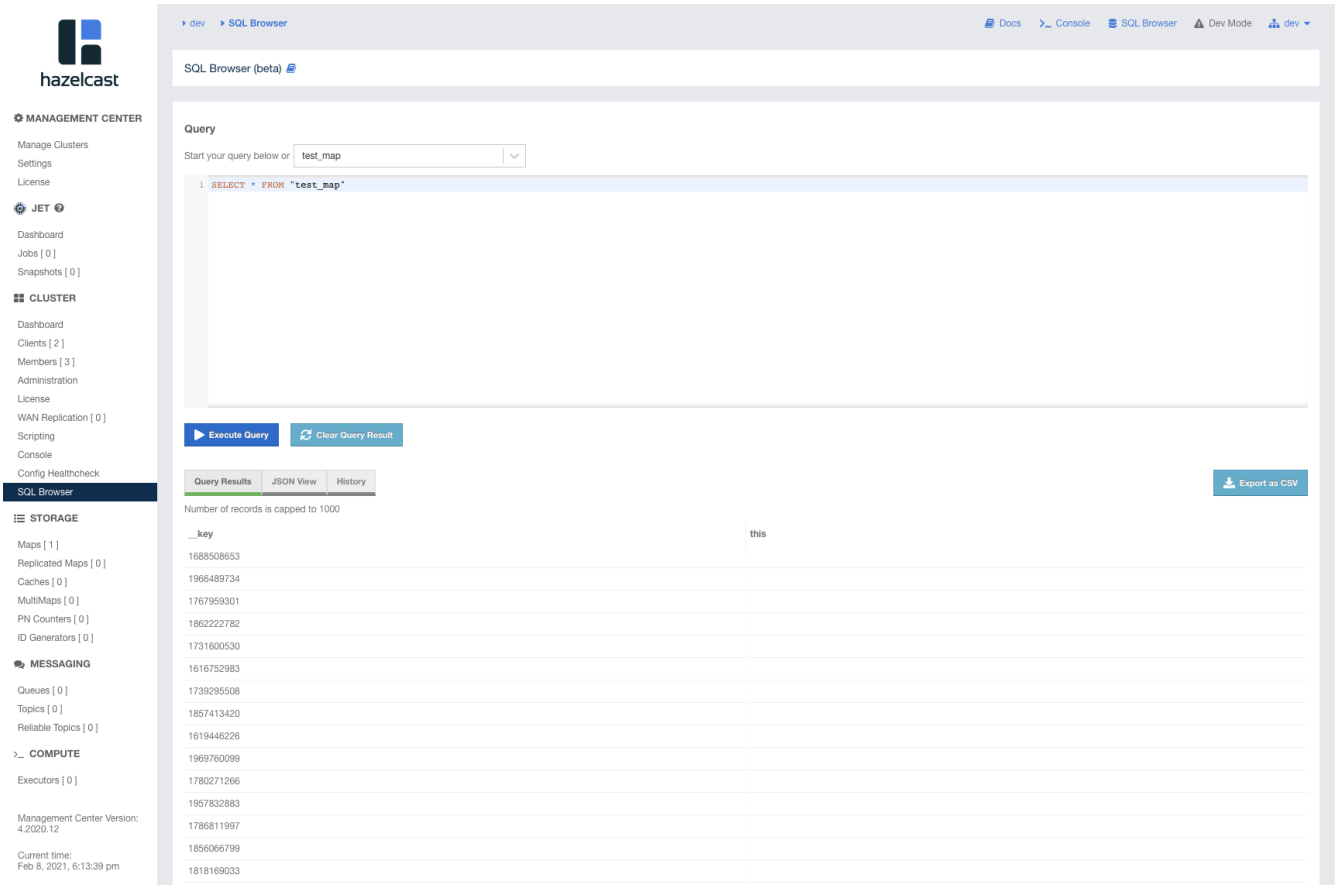



*Figure 6. Management Center SQL Browser*

Note: For running SQL queries in the beta version of this feature, it is mandatory for the client and server to have the same version. This means MC 4.2021.03 uses Hazelcast client 4.1 and can only run SQL queries on IMDG 4.1 alongside MC 4.2021.04 uses Hazelcast client 4.2 and therefore can

only run SQL queries on IMDG 4.2 even though it connects and shows the metrics of IMDG 4.1.

## 32.3. Data Structure and Member Management

The Caches, Maps, Queues, Topics, MultiMaps, and Executors pages each provide a drill-down view into the operational statistics of individual data structures. The Members page provides a drill-down view into the operational statistics of individual cluster members, including CPU and memory utilization, JVM Runtime statistics and properties, and member configuration. It also provides tools to run GC, take thread dumps, and shut down each member node.

## 32.4. Monitoring Cluster Health

The “Cluster Health” section on the Management Center homepage describes current backup and partition migration activity. While a member’s data is being backed up, the Management Center will show an alert indicating that the cluster is vulnerable to data loss if that node is removed from service before the backup is complete.

When a member node is removed from service, the cluster health section will show an alert while the data is re-partitioned across the cluster indicating that the cluster is vulnerable to data loss if any further nodes are removed from service before the re-partitioning is complete.

You may also set alerts to fire under specific conditions. In the “Alerts” tab, you can set alerts based on the state of cluster members as well as alerts based on the status of particular data types. For one or more members, and for one or more data structures of a given type on one or more members, you can set alerts to fire when certain watermarks are crossed.

When an alert fires, it will show up as an orange warning pane overlaid on the Management Center web interface.

Available member alert watermarks:

- Free memory has dipped below a given threshold
- Used heap memory has grown beyond a given threshold
- Number of active threads has dipped below a given threshold
- Number of daemon threads has grown above a given threshold

Available Map and MultiMap alert watermarks (greater than, less than, or equal to a given threshold):

- Entry count
- Entry memory size
- Backup entry count
- Backup entry memory size
- Dirty entry count
- Lock count

- Gets per second
- Average get latency
- Puts per second
- Average put latency
- Removes per second
- Average remove latency
- Events per second

Available Queue alert watermarks (greater than, less than, or equal to a given threshold):

- Item count
- Backup item count
- Maximum age
- Minimum age
- Average age
- Offers per second
- Polls per second

Available executor alert watermarks (greater than, less than, or equal to a given threshold):

- Pending task count
- Started task count
- Completed task count
- Average remove latency
- Average execution latency

## 32.5. Management Center Configuration Tool

The Management Center Configuration Tool (MC-Conf) is a command line tool that allows you to update certain parts of the Management Center configuration by using its built-in tasks. Mc-Conf script is delivered within the Management Center release package under the /bin/ folder.

You can perform the following tasks with MC-Conf tool,

- configure cluster connection
- create users and change user passwords
- configure LDAP Security Provider
- configure Active Directory Security Provider
- configure JAAS Security Provider
- reset Security Provider

- enable/disable Metrics Persistence

In order to see all available commands, run the MC-Conf script with no arguments as shown below.

```
./bin/mc-conf.sh
```

When you choose a specific command, you can see all tasks available for it. Please see example below.

```
bin % ./mc-conf.sh cluster
Usage: mc-conf cluster [-hV] TASK
Manage Cluster Connection Configs
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  add     Add Cluster Connection Config. Newly added cluster connection is
            automatically enabled.
          *Important notice* Make sure that Management Center web application
            is stopped (offline) before starting this task.

  remove  Remove Cluster Connection Config. *Important notice* Make sure that
            Management Center web application is stopped (offline) before
            starting this task.
```

Note: You must run the MC-Conf tool on the same machine where the Management Center web application is deployed.

Note: The Management Center must not be running when changes are made via MC-Conf.

Please see the related section of Management Center online documentation for more details.

## 32.6. Metrics Persistence

Metrics are <string,value> key-value pairs of data that capture the runtime information about the members and clients in a Hazelcast cluster. Metrics data includes various number of time series, such as CPU load, memory consumption, and operation counters.

Management Center collects the metrics used for building its view about the Hazelcast cluster from the metrics system. By default, Management Center persists the collected metrics on disk and stores metrics data for one day.

The Metrics Persistence toggle item on Management Center allows you to check the status of the cluster at a time in the past.

Please see the Metrics Persistence section of the Management Center online documentation for detailed information on the usage of Metrics Persistence feature and configuration options.

### 32.6.1. Management Center Home Directory

Management Center stores the metrics in a database file, under the Management Center home directory on the disk. The data files can be found in the <User's Home Directory>/hazelcast-mc/metrics directory, e.g., /home/someuser/hazelcast-mc/metrics. This directory can be changed using the *hazelcast.mc.home* property on the server where Management Center is running.

### 32.6.2. In-Memory Cache Size

When on-disk persistence is enabled, you may see the following warning messages in the Management Center log:

```
Could not find 10230 metrics time series out of 100500 during background persistence
run.
Consider increasing metrics in-memory cache size by setting `hazelcast.mc.metrics
.inMemory.max.size` system property.
```

This message indicates that Management Center could not persist some of the collected data points. That happens when the in-memory cache size is not enough to fit all metrics generated by your clusters and its size should be increased. You can increase it by using the *hazelcast.mc.metrics.inMemory.max.size* system property. This value is 300000 by default, which is enough to store up to 3 minutes of data for 100,000 metrics.

Note: Each in-memory cache entry takes around 0.5 KB of memory, so you may have to increase the maximum heap size in JVM when changing this setting to a greater value.

### 32.6.3. Configuring RocksDB temporary directory

Management Center uses the RocksDB native library under the hood for metrics persistence. By default, the RocksDB binary is extracted into the java.io.tmpdir directory. In case you run into any file system permission errors, you can override this default by setting the ROCKSDB_SHAREDLIB_DIR environment variable to the absolute path of an other directory. See the following example:

```
mkdir $HOME/tmp/rocksdb
export ROCKSDB_SHAREDLIB_DIR="$(realpath $HOME/tmp/rocksdb)"
java -jar hazelcast-management-center-4.2020.12.jar
```

## 32.7. Prometheus Exporter

Hazelcast Management Center can expose the metrics collected from cluster members to Prometheus. This feature can be turned on by setting the *hazelcast.mc.prometheusExporter.enabled* system property to **true**.

Prometheus can be configured to scrape Management Center in *prometheus.yml* as follows:

```
scrape_configs:
  - job_name: 'HZ MC'
    # scheme defaults to 'http'.
    static_configs:
    - targets: ['localhost:8080'] # replace this address with the network address of
Hazelcast Management Center
```

Note: If you want to visualize the Prometheus metrics using Grafana, then you can start with this dashboard.

Please see the related section of Management Center online documentation for more details.

# 32.8. Monitoring WAN Replication

You can also monitor WAN Replication process on Management Center. WAN Replication schemes are listed under the WAN menu item on the left. When you click on a scheme, a new tab for monitoring the targets that that scheme has appears on the right. In this tab you see a WAN Replication Operations Table for each target that belongs to this scheme. The following information can be monitored:

- Connected: Status of the member connection to the target.
- Events Published per Second: Number of published events per second. Please see the paragraph below.
- Average Event Latency: Average latency of sending a record to the target from this member.
- Outbound Queue: Number of records waiting in the queue to be sent to the target.
- Action: Stops/Resumes replication of this member's records.
- State: Shows current state of the WAN publisher on a member.

## 32.8.1. Synchronizing Clusters Dynamically with WAN Replication

Starting with Hazelcast IMDG version 3.8, you can use Management Center to synchronize multiple clusters with WAN Replication. You can start the sync process inside the WAN Sync interface of Management Center without any service interruption. Also in Hazelcast IMDG 3.8, you can add a new WAN Replication endpoint to a running cluster using Management Center. So at any time, you can create a new WAN replication destination and create a snapshot of your current cluster using the sync ability.

Hazelcast provides the following synchronization options:

1. Default WAN synchronization operation: It sends all the data of an IMap to a target cluster to align the state of target IMap with the source IMap.
2. WAN synchronization using Merkle trees.

Please use the "WAN Sync" screen of Management Center to display existing WAN replication configurations. You can use the "Add WAN Replication Config" button to add a new configuration, and the "Configure Wan Sync" button to start a new synchronization with the desired config.

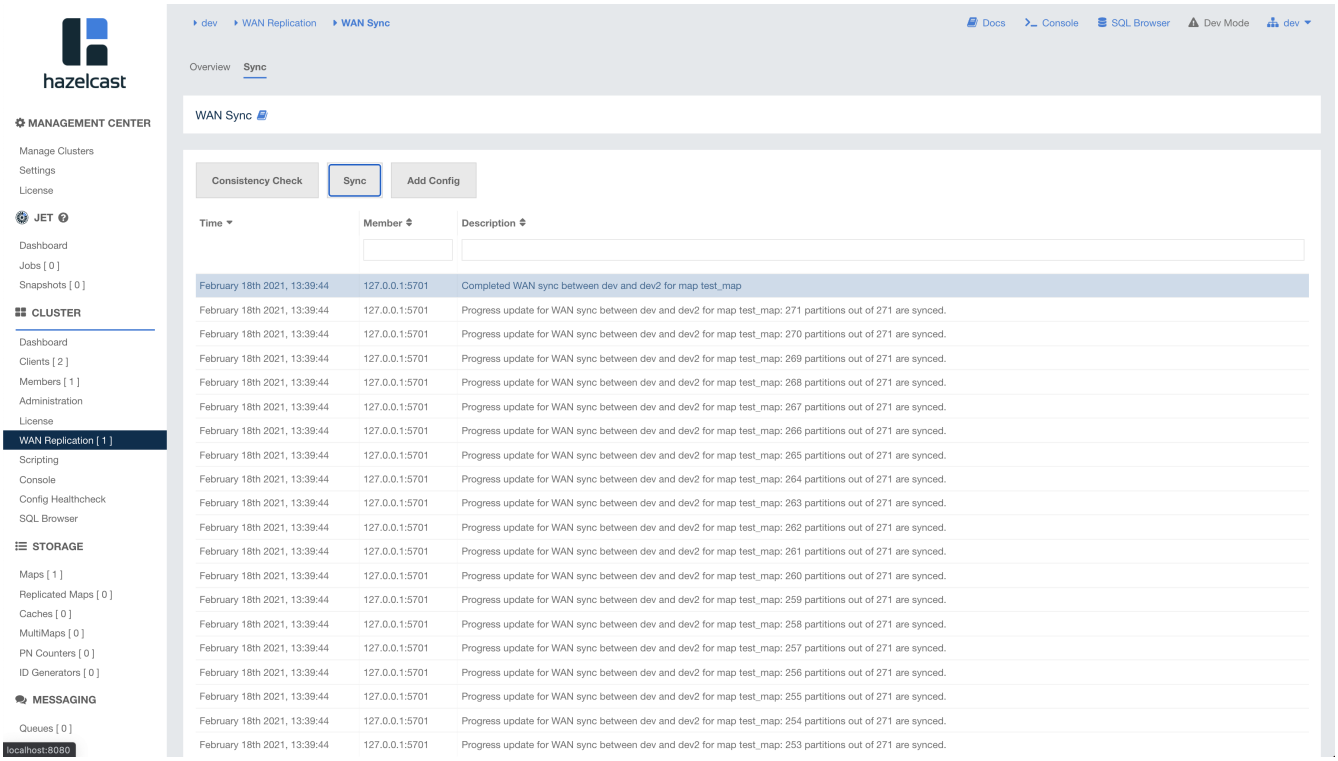



*Figure 7. Monitoring WAN Replication*

### 32.8.2. Delta WAN Synchronization

As it is mentioned above, Hazelcast has the default Full WAN synchronization feature, through which the maps in different clusters are synced by transferring all entries from the source to the target cluster. This may be not efficient since some of the entries have remained unchanged on both clusters and do not require to be transferred. Also, for the entries to be transferred, they need to be copied to on-heap on the source cluster. This may cause spikes in the heap usage, especially if using large off-heap stores.

In addition the default Full WAN synchronization, Hazelcast provides Delta WAN Synchronization which uses Merkle tree for the same purpose. It is a data structure used for efficient comparison of the difference in the contents of large data structures. The precision of this comparison is defined by Merkle tree's depth. Merkle tree hash exchanges can detect inconsistencies in the map data and synchronize only the entries which are different, instead of sending all the map entries.

Please see the related section in Reference Manual for more details.



As Hazelcast IMDG version 3.11, Delta WAN Synchronization is implemented only for Hazelcast IMap. It will also be implemented for ICache in the future releases.

## 32.9. Management Center Deployment

Management Center can be run directly from the command line, or it can be deployed on your Java application server/container. Please keep in mind that Management Center requires a license key to monitor clusters with more than 2 members, so make sure to provide your license key either as a startup parameter or from the user interface after starting Management Center.

Management Center has following capabilities in terms of security:

- JAAS based security feature, you can authenticate both cluster members and clients and perform access control checks on client operations.
- Intercepting socket connections before a member joins a cluster or a client connects to a member of a cluster that allows to add custom hooks to join and perform connection procedures (like identity checking using a custom network authentication protocol, etc.).
- Intercepting remote operation executed by the client.
- Encryption is based on Java Cryptography Architecture (JCA)
- Enabling TLS/SSL encrypts socket level communication between Hazelcast members and between Hazelcast clients and members.
- Security realms allow configuring JAAS authentication and/or own identity independently on the module which consumes this configuration.
- The Auditlog feature for observing some important cluster events.
- Active Directory Authentication.
- LDAP Authentication.

Please note that beginning with Management Center 3.10, Java 8 or above is required to run.

### 32.9.1. Limiting Disk Usage of Management Center

You can control the disk space used by the Management Center to avoid exceeding available disk space. That can be done by configuring the Metrics Persistence.

You can use the hazelcast.mc.metrics.disk.ttl.days system property to configure the Management Center's metrics persistence. This property specifies Time-to-Live (TTL) setting for each record in the metrics persistence. It is in days and its default value is 1.

It is important to understand that the TTL setting acts as a soft limit and gives you an indirect control over Management Center disk usage. The final disk usage depends on the volume of metrics persistence data generated by your clusters, i.e., the number of cluster members and data structures with enabled statistics that you have in the connected clusters.



You may want to decrease the TTL setting in situations when the Management Center uses too much disk space.

**Further Reading:**

- Management Center Product Information
- Online Documentation, Management Center
- Online Documentation, Clustered JMX Interface
- Online Documentation, Clustered REST Interface
- Online Documentation, Deploying and Starting

# 33. Enterprise Cluster Monitoring with JMX & REST (Subscription and Enterprise Feature)

Each Hazelcast IMDG node exposes a JMX management interface that includes statistics about distributed data structures and the state of that node's internals. The Management Center described above provides a centralized JMX and REST management API that collects all of the operational statistics for the entire cluster.

As an example of what you can achieve with JMX beans for an IMap, you may want to raise alerts when the latency of accessing the map increases beyond an expected watermark that you established in your load-testing efforts. This could also be the result of high load, long GC, or other potential problems that you might have already created alerts for, so consider the output of the following bean properties:

```
localTotalPutLatency
localTotalGetLatency
localTotalRemoveLatency
localMaxPutLatency
localMaxGetLatency
localMaxRemoveLatency
```

Similarly, you may also make use of the HazelcastInstance bean that exposes information about the current node and all other cluster members.

For example, you may use the following properties to raise appropriate alerts or for general monitoring:

- `memberCount`: If this is lower than the count of expected members in the cluster, raises an alert.
- `members`: Returns a list of all members connected in the cluster.
- `shutdown`: The shutdown hook for that node.
- `clientConnectionCount`: Returns the number of client connections. Raises an alert if lower than expected number of clients.
- `activeConnectionCount`: Total active connections.

**Further Reading:**

- Online Documentation, Monitoring With JMX
- Online Documentation, Clustered JMX
- Online Documentation, Clustered REST

## 33.1. Recommendations for Setting Alerts

We recommend setting alerts for at least the following incidents:

- CPU usage consistently over 90% for a specific time period
- Heap usage alerts:
  - Increasing old gen after every full GC while heap occupancy is below 80% should be treated as a moderate alert
  - Over 80% heap occupancy after a full GC should be treated as a red alert.
  - Too-frequent full GCs
- Node left event
- Node join event
- SEVERE or ERROR in Hazelcast IMDG logs

## 33.2. Actions and Remedies for Alerts

When an alert fires on a node, it's important to gather as much data about the ailing JVM as possible before shutting it down.

Logs: Collect Hazelcast IMDG server logs from all server instances. If running in a client-server topology, also collect client application logs before a restart.

Thread dumps: Make sure you take thread dumps of the ailing JVM using either the Management Center or jstack. Take multiple snapshots of thread dumps at 3 – 4 second intervals.

Heap dumps: Make sure you take heap dumps and histograms of the ailing JVM using jmap.

**Further Reading:**

- What to do in case of an OOME: http://blog.hazelcast.com/out-of-memory/
- What to do when one or more partitions become unbalanced (e.g., a partition becomes so large, it can't fit in memory): https://hazelcast.com/blog/controlled-partitioning/
- What to do when a queue store has reached its memory limit: http://blog.hazelcast.com/overflow-queue-store/
- http://blog.scoutapp.com/articles/2009/07/31/understanding-load-averages
- http://docs.oracle.com/javase/7/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html

# 34. Handling Network Partitions

In an ideal world the network is always fully up or fully down. However, in reality, network partitions can happen. This chapter discusses how to handle those rare cases.

## 34.1. Split-Brain on Network Partition

In certain cases of network failure, some cluster members may become unreachable. These members may still be fully operational. They may be able to see some, but not all, other extant cluster members. From the perspective of each node, the unreachable members will appear to have gone offline. Under these circumstances, what was once a single cluster will divide into two or more clusters. This is known as network partitioning, or "Split-Brain Syndrome".

Consider a five-node cluster as depicted in below figure:

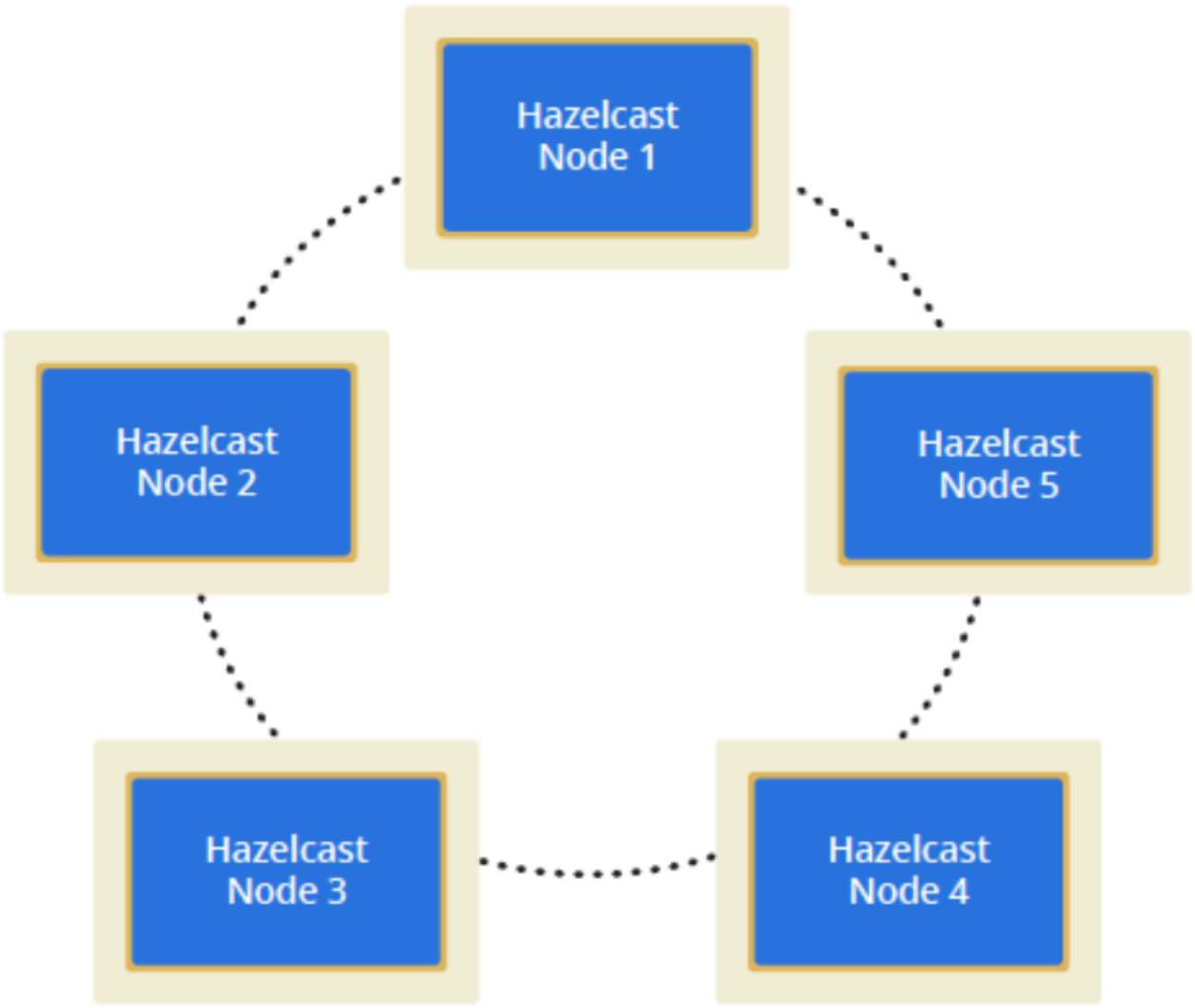



*Figure 8. Five-Node Cluster*

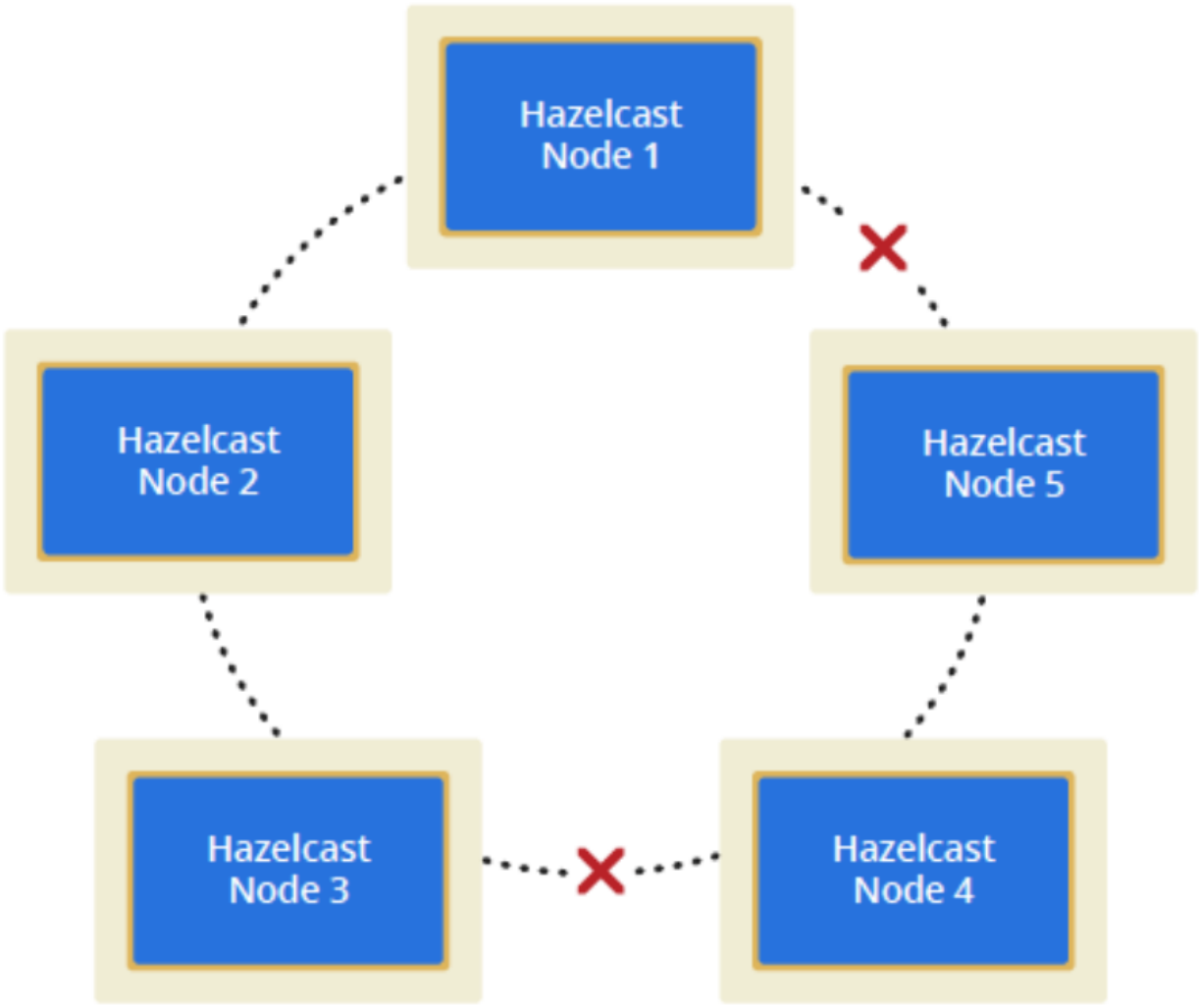



*Figure 9. Network failure isolates nodes one, two, and three from nodes four and five*

All five nodes have working network connections to each other and respond to health check heartbeat pings. If a network failure causes communication to fail between nodes four and five and the rest of the cluster (Figure 6), from the perspective of nodes one, two, and three, nodes four and five will appear to have gone offline. However, from the perspective of nodes four and five, the opposite is true: nodes one through three appear to have gone offline (Figure 7).

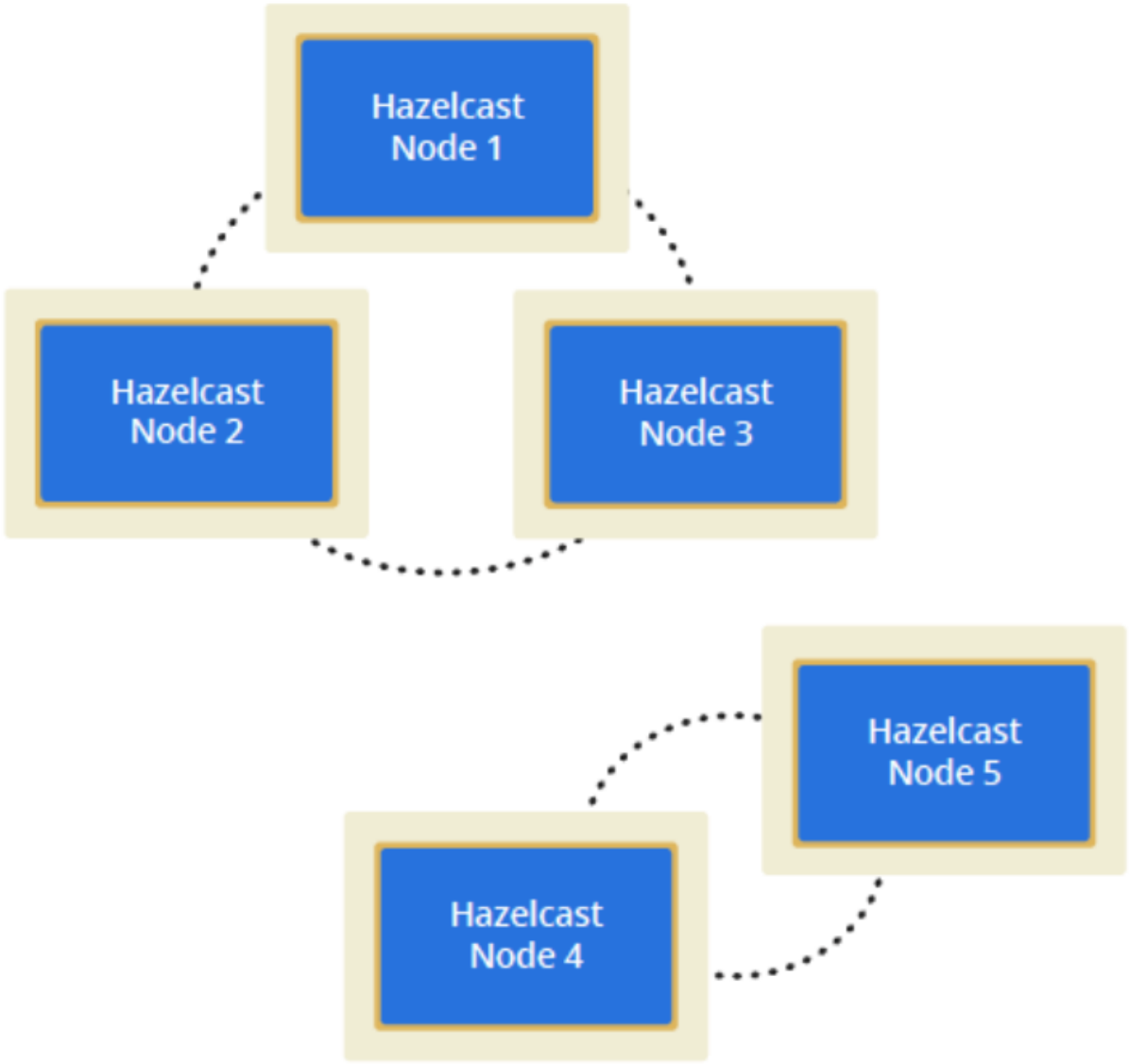



*Figure 10. Split-Brain*

How should you respond to a split-brain scenario? The answer depends on whether consistency of data or availability of your application is of primary concern. In either case, because a split-brain scenario is caused by a network failure, you must initiate an effort to identify and correct the network failure. Your cluster cannot be brought back to a steady state until the underlying network failure is fixed.

If availability is of primary concern, especially if there is little danger of data becoming inconsistent across clusters (e.g., you have a primarily read-only caching use case) then you may keep both clusters running until the network failure has been fixed. Alternately, if data consistency is of primary concern, it may make sense to remove the clusters from service until the split-brain is repaired. If consistency is your primary concern, use Split-Brain Protection as discussed below.

## 34.2. Split-Brain Protection

Split-brain Protection provides the ability to prevent the smaller cluster in a split-brain from being used by your application where consistency is the primary concern.

This is achieved by defining and configuring a split-brain protection. The minimum cluster size is required for operations to occur.

Tip: It is preferable to have an odd-sized initial cluster size to prevent a single network partition from creating two equal sized clusters.

So imagine we have a 9 node cluster. The minimum cluster size is configured as 5. If any split-brains occur, the smaller clusters of sizes 1, 2, 3, 4 will be prevented from being used. Only the larger cluster of size 5 will be allowed to be used. The following declaration would be added to the Hazelcast IMDG configuration:

```
<split-brain-protection name="splitBrainProtectionRuleWithFiveMembers" enabled="true">
    <minimum-cluster-size>5</minimum-cluster-size>
</split-brain-protection>
```

Attempts to perform operations against the smaller cluster will be rejected and the rejected operations will return a `SplitBrainProtectionException` to their callers. Write operations, Read operations or both can be configured with split-brain protection.

Your application will continue normal processing on the larger remaining cluster. Any application instances connected to the smaller cluster will receive exceptions which, depending on the programming and monitoring setup, should throw alerts. The key point is that rather than applications continuing in error with stale data, they are prevented from doing so.

Note: Both in the API/code samples and documentation, the term "quorum" has been replaced by "split-brain protection" with Hazelcast 4.x.

### 34.2.1. Time Window

Cluster Membership is established and maintained by heart-beating. A network partition will present itself as some members being unreachable. While configurable, it is normally seconds or tens of seconds before the cluster is adjusted to exclude unreachable members. The cluster size is based on the currently understood number of members.

For this reason there will be a time window between the network partition and the application of split-brain protection. The length of this window will depend on the failure detector. Every member will eventually detect the failed members and will reject the operation on the data structure that requires the minimum-cluster-size.

Split-brain protection, since it was introduced, has relied on the observed count of cluster members as determined by the member's cluster membership manager. Starting with Hazelcast 4.x, split-brain protection can be configured with new out-of-the-box `SplitBrainProtectionException` implementations which determine the presence of minimum-cluster-size independently of the cluster membership manager, taking advantage of heartbeat, ICMP, and other failure-detection information configured on Hazelcast members.

In addition to the Member Count Split-Brain Protection, the two built-in split-brain protection functions are as follows:

1. *Probabilistic Split-Brain Protection Function*: Uses a private instance of `PhiAccrualClusterFailureDetector`, which is updated with member heartbeats, and its parameters can be fine-tuned to determine live members, separately from the cluster's membership manager. This function is configured by adding the `probabilistic-split-brain-protection` element to the split-brain protection configuration.
2. *Recently Active Split-Brain Protection Function*: Can be used to implement more conservative split-brain protection by requiring that a heartbeat has been received from each member within a configurable time window. This function is configured by adding the `recently-active-split-brain-protection` element to the split-brain protection configuration.

You can also implement your own custom split-brain protection function by implementing the `SplitBrainProtectionFunction` interface.

Please see the reference manual for more details regarding the configuration.

### 34.2.2. Protected Data Structures

The following data structures are protected:

- Map (3.5 and higher)
- Map (High-Density Memory Store backed) (3.10 and higher)
- Transactional Map (3.5 and higher)
- Cache (3.5 and higher)
- Cache (High-Density Memory Store backed) (3.10 and higher)
- Lock (3.8 and higher)
- Queue (3.8 and higher)
- IExecutorService, DurableExecutorService, IScheduledExecutorService, MultiMap, ISet, IList, Ringbuffer, Replicated Map, Cardinality Estimator, IAtomicLong, IAtomicReference, ISemaphore, ICountdownLatch (3.10 and higher)

Each data structure to be protected should have the split-brain protection configuration added to it.

**Further Reading:**

- Online Documentation, Split-Brain Protection

## 34.3. Split-Brain Resolution

Once the network is repaired, the multiple clusters must be merged back together into a single cluster. This normally happens by default and the multiple sub-clusters created by the split-brain merge again to re-form the original cluster. This is how Hazelcast IMDG resolves the split-brain condition:

1. Checks whether sub-clusters are suitable to merge.
    a. Sub-clusters should have compatible configurations: same group name and password, same partition count, same joiner types etc.

b. Sub-clusters' membership intersection set should be empty; they should not have common members. If they have common members, that means there is a partial split; sub-clusters postpone the merge process until membership conflicts are resolved.

c. Cluster states of sub-clusters should be ACTIVE.

2. Performs an election to determine the winning cluster. Losing side merges into the winning cluster.

   a. The bigger sub-cluster, in terms of member count, is chosen as winner and smaller one merges into the bigger.

   b. If sub-clusters have an equal number of members, a pure function with two sub-clusters given as input is executed to determine the winner on both sides. Since this function produces the same output with the same inputs, the winner can be consistently determined by both sides.

3. After the election, Hazelcast IMDG uses merge policies for supported data structures to resolve data conflicts between split clusters. A merge policy is a callback function to resolve conflicts between the existing and merging records. Hazelcast IMDG provides an interface to be implemented and also a few built-in policies ready to use.

Starting with Hazelcast IMDG version 3.10, all merge policies are implementing the unified interface `com.hazelcast.spi.SplitBrainMergePolicy`. We provide the following out-of-the-box implementations:

- **DiscardMergePolicy:** the entry from the smaller cluster will be discarded.
- **ExpirationTimeMergePolicy:** the entry with the higher expiration time wins.
- **HigherHitsMergePolicy:** the entry with the higher number of hits wins.
- **HyperLogLogMergePolicy:** specialized merge policy for the CardinalityEstimator, which uses the default merge algorithm from HyperLogLog research, keeping the max register value of the two given instances.
- **LatestAccessMergePolicy:** the entry with the latest access wins.
- **LatestUpdateMergePolicy:** the entry with the latest update wins.
- **PassThroughMergePolicy:** the entry from the smaller cluster wins.
- **PutIfAbsentMergePolicy:** the entry from the smaller cluster wins if it doesn't exist in the cluster.

The statistic based out-of-the-box merge policies are supported by `IMap`, `ICache`, `ReplicatedMap` and `MultiMap`. The `HyperLogLogMergePolicy` is just supported by the `CardinalityEstimator`.

Please see the reference manual for details.

**Further Reading:**

- Online Documentation, Network Partitioning

# 35. How to Report Issues to Hazelcast

## 35.1. Hazelcast Support Subscribers

A support subscription from Hazelcast will allow you to get the most value out of your selection of Hazelcast IMDG. Our customers benefit from rapid response times to technical support inquiries, access to critical software patches, and other services which will help you achieve increased productivity and quality.

Learn more about Hazelcast support subscriptions: https://hazelcast.com/pricing/

### 35.1.1. Opening a Ticket

If your organization subscribes to Hazelcast Support, and you already have an account setup, you can login to your account and open a support request using our ticketing system: https://hazelcast.zendesk.com/

When submitting a ticket to Hazelcast, please provide as much information and data as possible:

### 35.1.2. Prepare to Open a support Ticket

- Make sure that all your environments are capturing Hazelcast diagnostics logs. This a primary on diagnosing issues with Hazelcast environments. * If your environments are not capturing diagnostics logs, please update them to capture diagnostics logs.
- Make note of your issue with a clear description of the issue for a title text. This will allow Support to route the issue to the proper expert
- Make a note of the steps to reproduce if possible. If not please capture the sequence of events that led to the problem.
- Write a complete description of the problem along with any error found.
- Capture any relevant screen shots and or errors noted.
- Create a support ticket on Hazelcast Zendesk support Portal
- Attach the appropriate severity to the ticket.
- PROD issues that affect Production should be considered for Severity 1
- All other issues in other environments should be considered severity 2 or 3 depending on urgency.
- Dev issue should be considered Severity 3 and priority Low
- All other issues e.g. questions or documentation review should be considered severity 3 or higher.

### 35.1.3. Adding Details to the support Ticket

- When you open a support ticket add a concise title and description of the problem.
- Add steps to reproduce as best as you can document them so that support can attempt to

reproduce the problem. This includes Detailed description of incident – what happened and when.

- Reproducible test case (optional: Hazelcast engineering may ask for it if required)
- Details of use case. This is crucial as it helps support narrow down the features and functionality in play when the problem occured.
- Attach any specific error found.
- Attach the complete logs files i.e. Hazelcast IMDG logs
- Attach Hazelcast Process logs
- Attach Hazelcast Health Monitor logs
- Thread dumps from all server nodes
- Heap dumps
- Networking logs
- Time of incident
- When providing Hazelcast logs please make sure that the system and environment details that are captured at system startup are included. Even if you truncate the logs. Hazelcast Diagnostic logs
- Please do not truncate diagnostics logs. They only capture Hazelcast systems specific information and details.
- Please make sure that the logs capture data around the date and time of the incident.

Please follow these steps for prompt help with support. Please note that the more information is provided upfront the better and after support should be able to help. When providing

Lastly be prompt in your communication with Hazelcast support, helps to ensure timely resolution of issues.

### 35.1.4. Support SLA

SLAs may vary depending upon your subscription level. If you have questions about your SLA, please refer to your support agreement, your "Welcome to Hazelcast Support" email, or open a ticket and ask. We'll be happy to help.

## 35.2. Hazelcast IMDG Open Source Users

Hazelcast has an active open source community of developers and users. If you are a Hazelcast IMDG open source user, you will find a wealth of information and a forum for discussing issues with Hazelcast developers and other users:

- Hazelcast Google Group - https://groups.google.com/forum/#!forum/hazelcast
- Stack Overflow - http://stackoverflow.com/questions/tagged/hazelcast

You may also file and review issues on the Hazelcast IMDG issue tracker on GitHub: https://github.com/hazelcast/hazelcast/issues

To see all of the resources available to the Hazelcast community, please visit the community page on Hazelcast.org.