hazelcast

Developing low latency applications with Hazelcast

Sharath Sahadevan
June 11, 2020

Our Mission: Distributed Computing. Simplified. Two In-Memory Products

HAZELCAST IMDG is an operational, in-memory, distributed computing platform that manages data using in-memory storage, and performs parallel execution for breakthrough application speed and scale.

HAZELCAST JET is the ultra fast, application embeddable, 3rd generation stream processing engine for low latency batch and stream processing.

An In-Memory Data Grid Is…

A data model distributed across many servers in a single or across multiple data centers and/or clouds

- All data is stored in the RAM of the servers
- Servers can be added or removed to increase RAM
- Data model is non-relational and is object-based
- Applications written in Java, C++, C#/.NET, Node.js, Python, Golang and Scala can receive benefits of distributed data store
- Resilient, allowing non-disruptive detection and recovery of a single or multiple servers

Why Hazelcast IMDG?

Scale-out computing enables cluster capacity to be increased or decreased on-demand

Resilience with automatic recovery from member failures without losing data, while minimizing performance impact on running applications

Simple Programming Model provides a way for developers to easily program a cluster application as if it is a single process

Fast Application Performance enables very large data sets to be held in main memory for real-time performance, featuring support for Intel Optane DC Persistent Memory

intel OPTANE™ DC PERSISTENT MEMORY

hazelcast

Hazelcast IMDG Key Differentiators

Comprehensive

- Very rich APIs and features driven by our huge user base
- Features for simple caching use cases at scale
- Features for rich IMDG use cases for Financial Services and OEM
- Quick response to new feature requests
- Hazelcast IMDG has a solution to meet your needs
- A wide selection of client languages: Java, C++, C#, Python, Scala, and Node.js

Productive

- Legendary ease of use
- Uses standard interfaces to make learning easy
- Very wide usage means more developers know Hazelcast
- Embedded mode has no operational management overhead
- Scaling, failure handling and recovery are all automated for ease of operational management

Performant

- Fully in-memory store transforms and ingests data in microseconds, providing throughput and query
- Both a distributed computation system and a data store in the same footprint — eliminates network transit latency penalties from data movements
- Fastest distributed cache

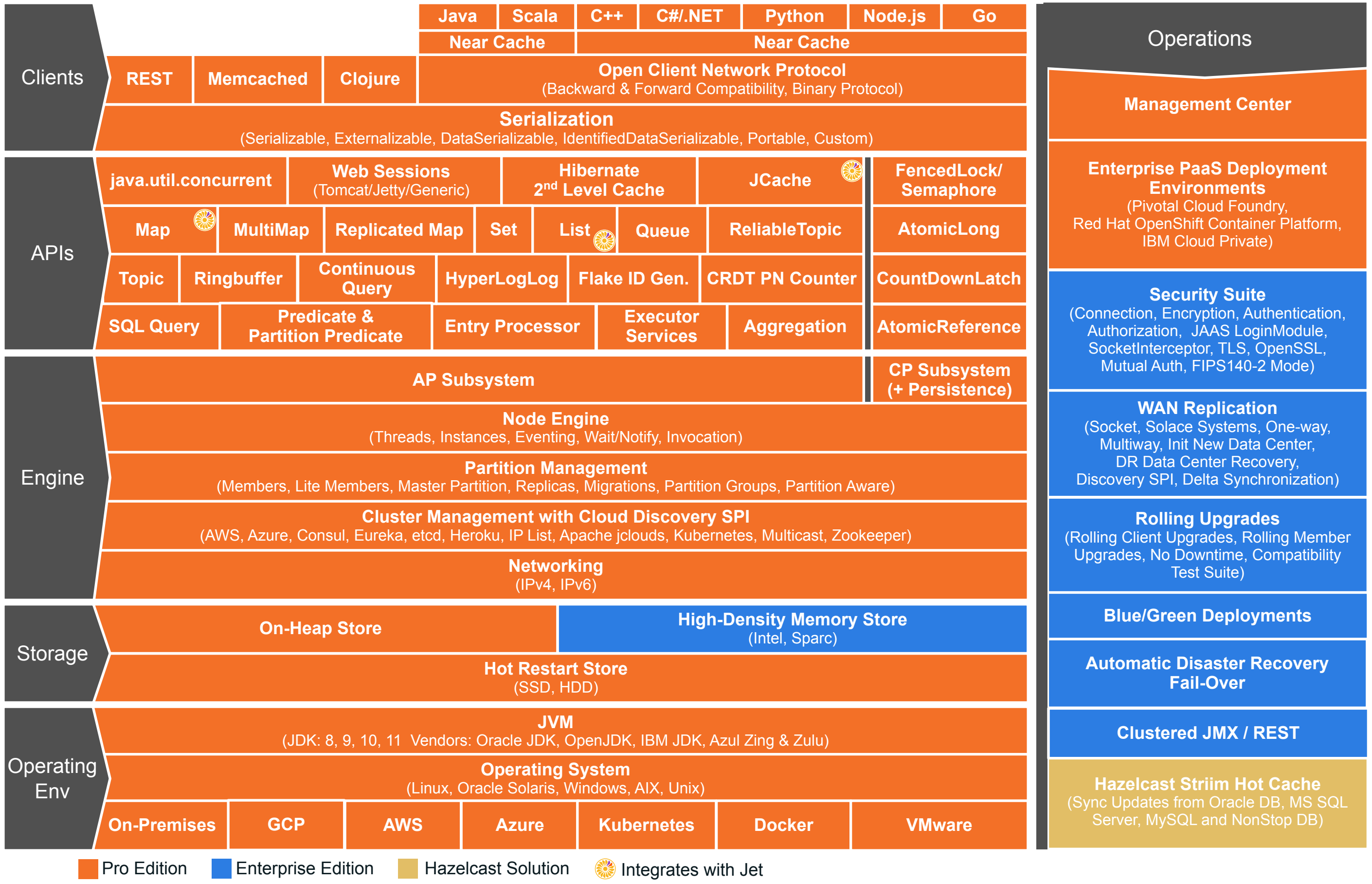
Enterprise-grade

- Stable at vertical and horizontal scale
- No maintenance windows required for upgrade or new application deployment
- Cloud and container ready: Pivotal Cloud Foundry, OpenShift, Docker, Azure, AWS, Kubernetes, …
- Industry-leading security
- Most widely deployed IMDG, with tens of thousands of deployments in production worldwide

hazelcast

Hazelcast IMDG
Product Overview

Hazelcast In-Memory Computing Platform



Clients
- Java | Scala | C++ | C#/.NET | Python | Node.js | Go
- Near Cache (Java, Scala) | Near Cache (C++, C#/.NET, Python, Node.js, Go)
- REST | Memcached | Clojure | **Open Client Network Protocol** (Backward & Forward Compatibility, Binary Protocol)
- **Serialization** (Serializable, Externalizable, DataSerializable, IdentifiedDataSerializable, Portable, Custom)

APIs
- java.util.concurrent | Web Sessions (Tomcat/Jetty/Generic) | Hibernate 2nd Level Cache | JCache 🟡 | FencedLock/ Semaphore
- Map 🟡 | MultiMap | Replicated Map | Set | List 🟡 | Queue | ReliableTopic | AtomicLong
- Topic | Ringbuffer | Continuous Query | HyperLogLog | Flake ID Gen. | CRDT PN Counter | CountDownLatch
- SQL Query | Predicate & Partition Predicate | Entry Processor | Executor Services | Aggregation | AtomicReference

Engine
- AP Subsystem | CP Subsystem (+ Persistence)
- **Node Engine** (Threads, Instances, Eventing, Wait/Notify, Invocation)
- **Partition Management** (Members, Lite Members, Master Partition, Replicas, Migrations, Partition Groups, Partition Aware)
- **Cluster Management with Cloud Discovery SPI** (AWS, Azure, Consul, Eureka, etcd, Heroku, IP List, Apache jclouds, Kubernetes, Multicast, Zookeeper)
- **Networking** (IPv4, IPv6)

Storage
- On-Heap Store | **High-Density Memory Store** (Intel, Sparc)
- **Hot Restart Store** (SSD, HDD)

Operating Env
- **JVM** (JDK: 8, 9, 10, 11 Vendors: Oracle JDK, OpenJDK, IBM JDK, Azul Zing & Zulu)
- **Operating System** (Linux, Oracle Solaris, Windows, AIX, Unix)
- On-Premises | GCP | AWS | Azure | Kubernetes | Docker | VMware

Operations
- **Management Center**
- **Enterprise PaaS Deployment Environments** (Pivotal Cloud Foundry, Red Hat OpenShift Container Platform, IBM Cloud Private)
- **Security Suite** (Connection, Encryption, Authentication, Authorization, JAAS LoginModule, SocketInterceptor, TLS, OpenSSL, Mutual Auth, FIPS140-2 Mode)
- **WAN Replication** (Socket, Solace Systems, One-way, Multiway, Init New Data Center, DR Data Center Recovery, Discovery SPI, Delta Synchronization)
- **Rolling Upgrades** (Rolling Client Upgrades, Rolling Member Upgrades, No Downtime, Compatibility Test Suite)
- **Blue/Green Deployments**
- **Automatic Disaster Recovery Fail-Over**
- **Clustered JMX / REST**
- **Hazelcast Striim Hot Cache** (Sync Updates from Oracle DB, MS SQL Server, MySQL and NonStop DB)

🟧 Pro Edition 🟦 Enterprise Edition 🟨 Hazelcast Solution 🟡 Integrates with Jet

Hazelcast IMDG Subscriptions?
What Is the Right Feature Set?

	Enterprise Edition	Pro Edition
	For mission-critical, high-performance scale-out and scale-up of your applications	For service-level guaranteed operation of your application
Management Center	✓	✓
MC Packs: Notification, Security, Support, Integration	✓	✓
MC Packs: Config	✓	
Enterprise PaaS Deployments (OpenShift, Pivotal Cloud Foundry)	✓	✓
Hot Restart Store	✓	✓
CP Subsystem Persistence	✓	✓
24/7 Support	1-hour SLA	4-hour SLA
Security Suite	✓	
WAN Replication	✓	
High-Density Memory Store	✓	
Rolling Upgrades – No Service Interruption	✓	
Blue/Green Deployments	✓	
Automatic Disaster Recovery Fail-Over	✓	

hazelcast

Technical Use Cases

Technical Use Cases

Caching

- High-Density Memory Store, client and member
- Full JCache support
- Elastic scalability
- Super fast
- High availability
- Fault tolerant
- Cloud ready

In-Memory Data Grid

- Simple, modern APIs
- Distributed data structures
- Distributed compute
- Distributed clustering
- Object-oriented and non-relational
- Elastic and scalable
- Transparent database integration
- Client-server and/or embedded architecture

Microservices Infrastructure

- Isolation of Services with many, small clusters for easier troubleshooting & maintenance
- Service registry
- Multiple network discovery mechanisms
- Inter-process messaging
- Fully embeddable
- Resilient and flexible

Web Session Clustering

- Seamless failover between user sessions
- High performance
- No application alteration
- Easy scale-out
- Fast session access
- Offload to existing cluster
- Tomcat, Jetty + any Web Container
- Works efficiently with large session objects using delta updates

hazelcast

Technical Use Cases: Caching

- High-Density Memory Store, client and member
- Full JCache support
- Elastic scalability
- Super fast
- High availability
- Fault tolerance
- Cloud readiness

High-Density Caching

Cache-as-a-Service (CaaS)

JCache Provider

Database Caching

Hibernate Second Level Cache

Software AG Terracotta Replacement

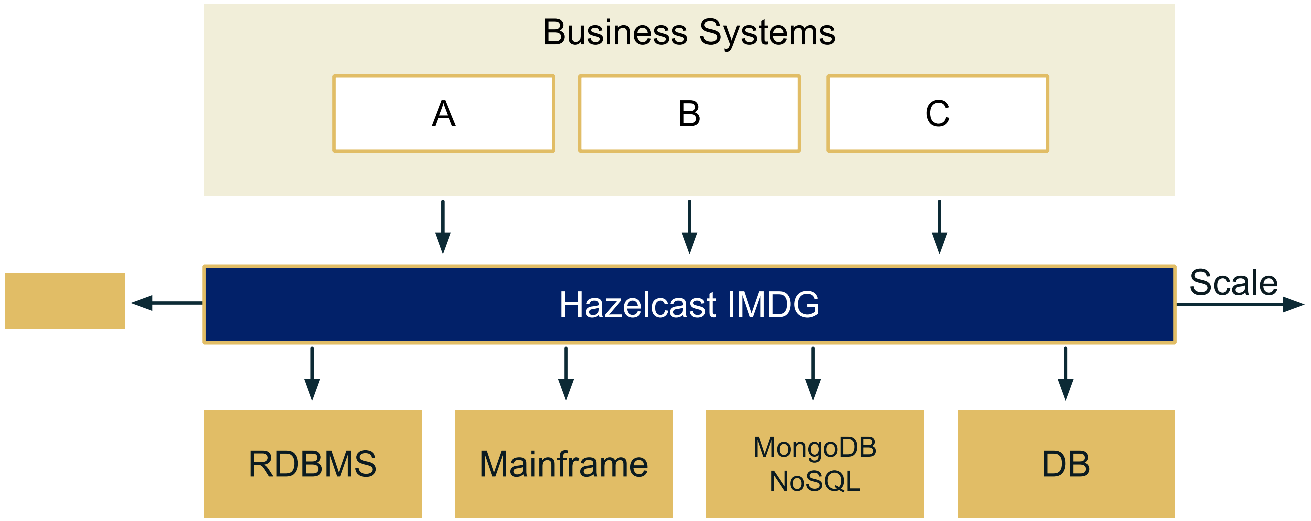
Application Scaling

Elastic Memcached

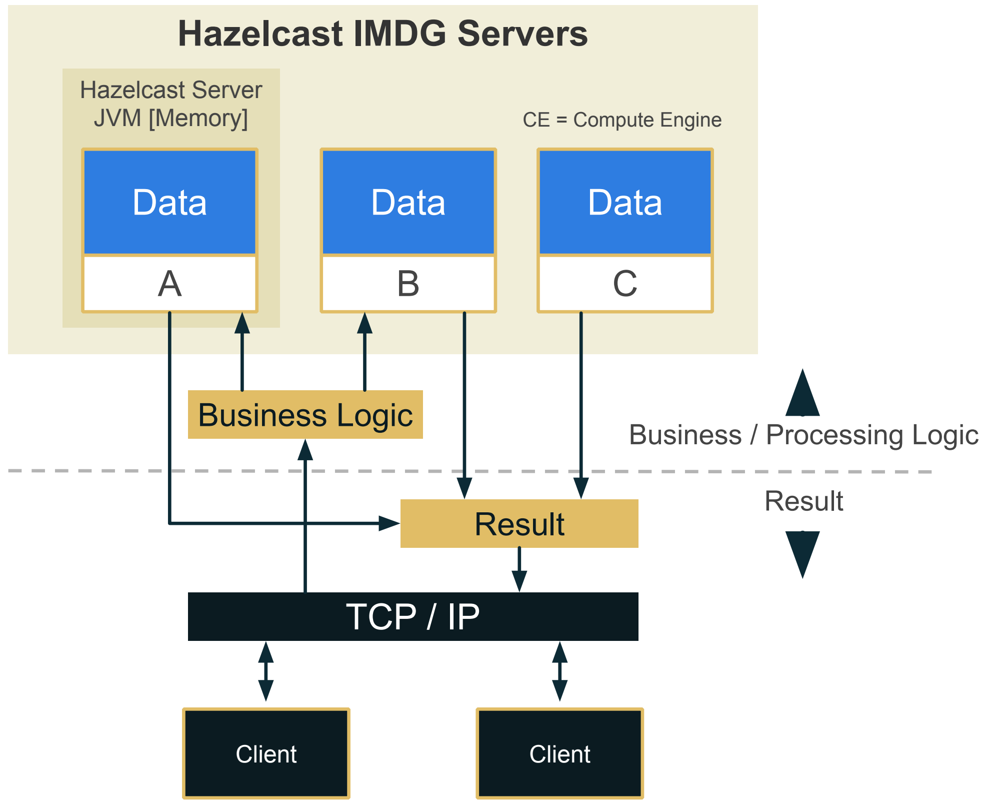
Spring Cache

- In-memory access to frequently used data across an elastically-scalable grid
- Provide 100's of GB of near cached data to applications for massive scalability
- Dynamically cluster/pool memory and processors

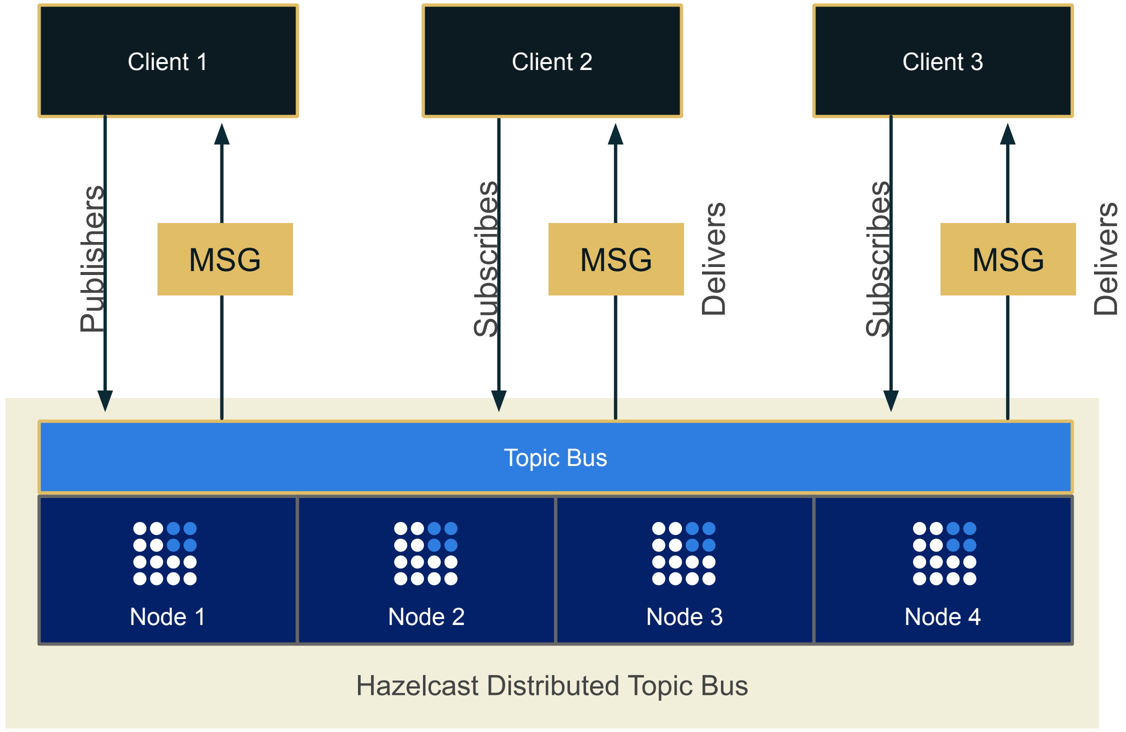
Technical Use Cases: Cache in Front of a Data Store



Technical Use Cases: In-Memory Data Grid Compute



Technical Use Cases: In-Memory Data Grid Messaging



Technical Use Cases: Microservices Infrastructure

- Reduce restoration time after failure
- Ease of adding new functionality without affecting users
- Ease-of-use on a developer machine
- Modular structure for larger dev teams
- Ease-of-deployment, simple services
- Flexibility of diverse technological landscape

Network Discovery

Service Registry

Service Discovery

Inter-process Communication

Storage & Isolation

- Library structure simple to use, more efficient development process than frameworks
- Standard JDK Collections such as Lists, Sets, Queues and constructs such as Topics and Ringbuffers facilitate inter-process messaging
- Clients for several languages (currently Java, C#, C/C++, Python, Node.js and Scala) and support for a wide variety of serialization choices
- Shared or distributed data store intermediated via service requests to the data grid
- Elastically-scalable memory for each independent service

hazelcast

Technical Use Cases: Microservices Infrastructure Easier, More Scalable Path to Microservices

Simple and configurable backbone that makes moving an application to a distributed environment a lot easier

Library that is simpler to use and understand, making the development process more efficient

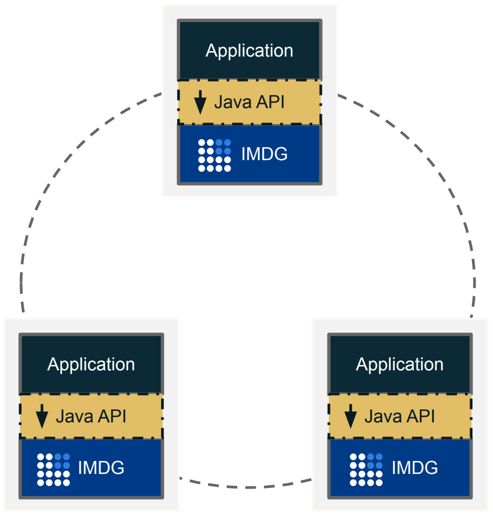
Hazelcast IMDG
is a…

Highly scalable In-Memory Data Grid (e.g. IMap and JCache), and supports other standard JDK Collections such as Lists, Sets, Queues and some other constructs such as Topics and Ringbuffers that can be easily leveraged for inter-process messaging

EE PS

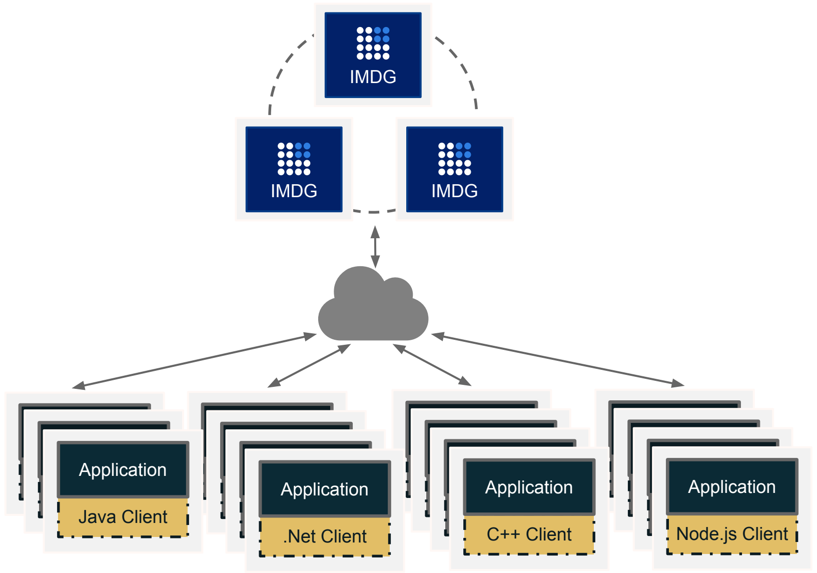
Hazelcast IMDG Deployment Options

Embedded Mode



Great for microservices,
OEM and ops simplification

Client-Server Mode



Great for scale-up or scale-out deployments with cluster lifecycle decoupled from app servers

Clients available in Java, Node.js, C#/.NET, C++, Scala, Python, Clojure and Golang

hazelcast

Technical Use Cases: Web Session Clustering

- High performance
- No application alteration
- Easy scale-out
- Fast session access
- Off load to existing cluster
- Tomcat, Jetty and Generic

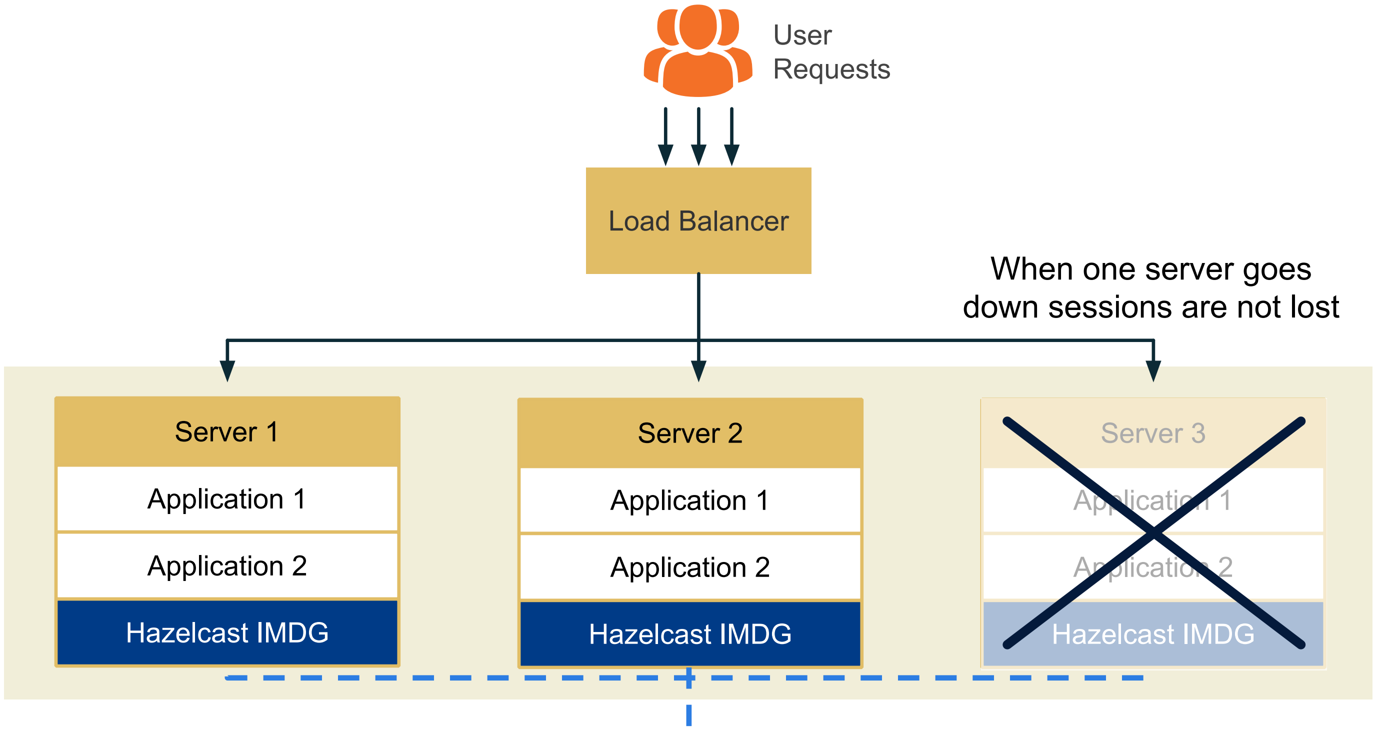
Tomcat Web Session Replication

Jetty Web Session Replication

Generic Web Session Replication

- No lost user sessions with even distribution across Hazelcast IMDG structure
- Automatic session failover

Technical Use Cases: Web Session Clustering



Sessions are stored in Hazelcast

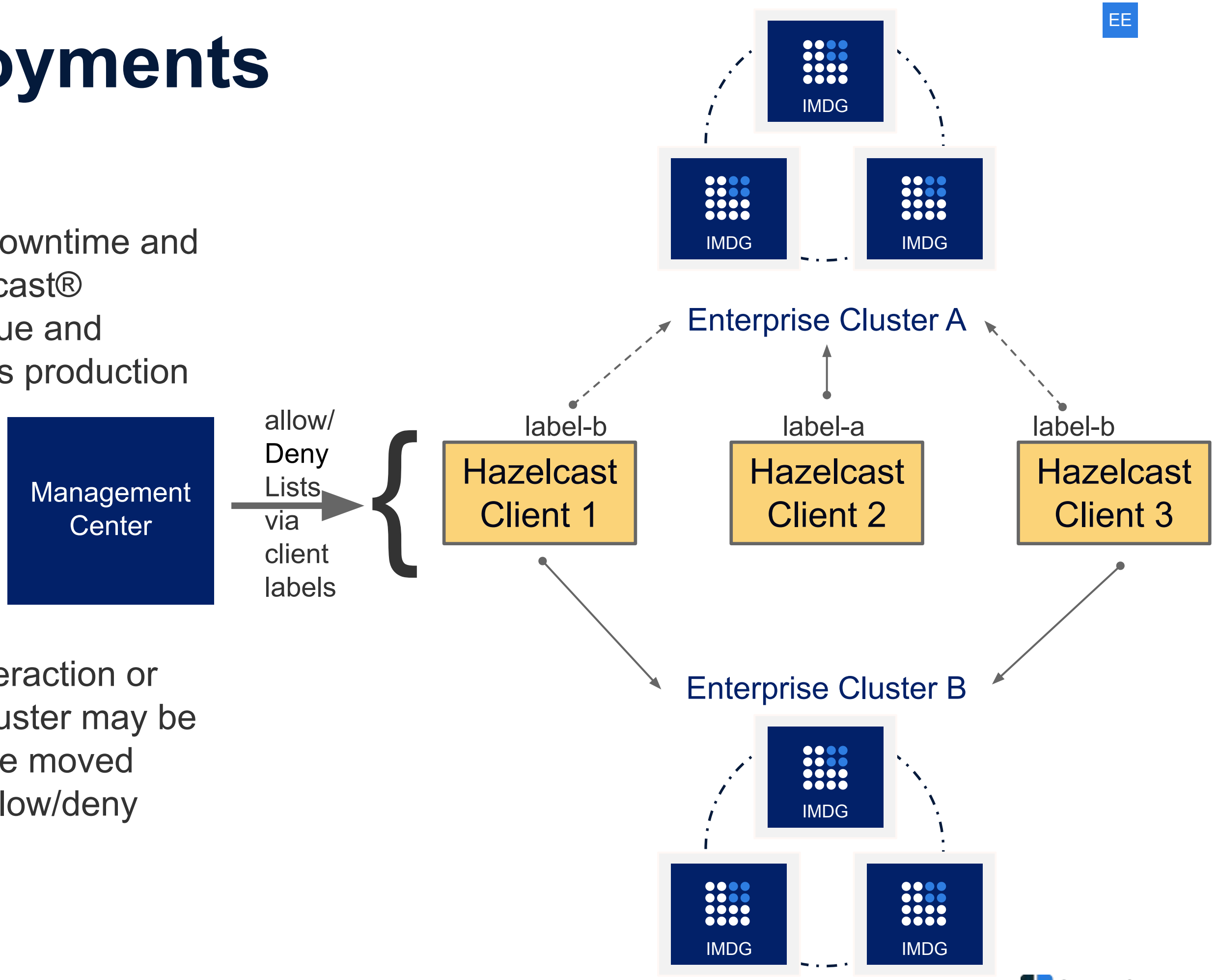
hazelcast

Blue/Green Deployments & Automatic Disaster Recovery Fail-Over

Blue/Green Deployments

EE

Blue/Green Deployments reduce downtime and risk by running two identical Hazelcast® Enterprise IMDG clusters called Blue and Green. One of the clusters provides production services to clients whilst the other cluster can be upgraded with new application code. Hazelcast Blue/Green Deployment functionality allows clients to be migrated from one cluster to another without client interaction or cluster downtime. All clients of a cluster may be migrated or groups of clients can be moved with the use of label filtering and allow/deny lists.

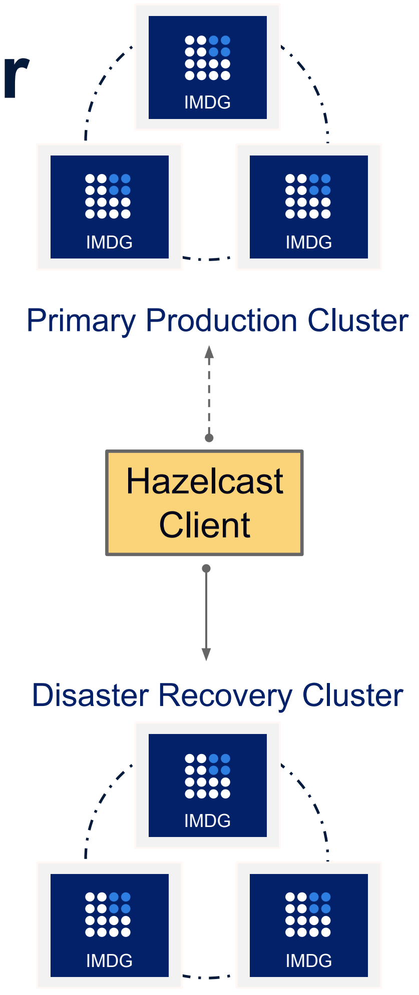


hazelcast

Automatic Disaster Recovery Failover

EE

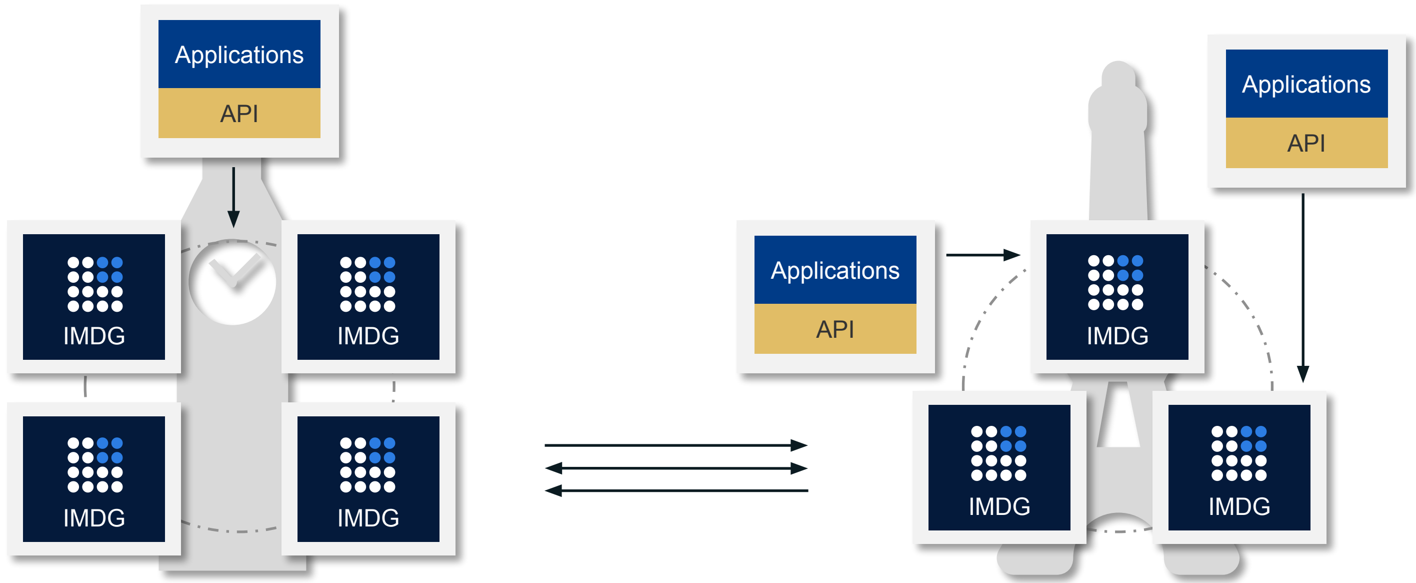
Automatic disaster recovery fail-over provides clients connected to Hazelcast® Enterprise IMDG clusters with the ability to automatically fail-over to a Disaster Recovery cluster should there be an unplanned outage of the primary production Hazelcast cluster.



hazelcast

EE

WAN Replication: Deployment Options



Independent clusters, sharing selected data to maintain alignment

One-way or two-way

Choose which data to share, select by type, can de-select records by attributes

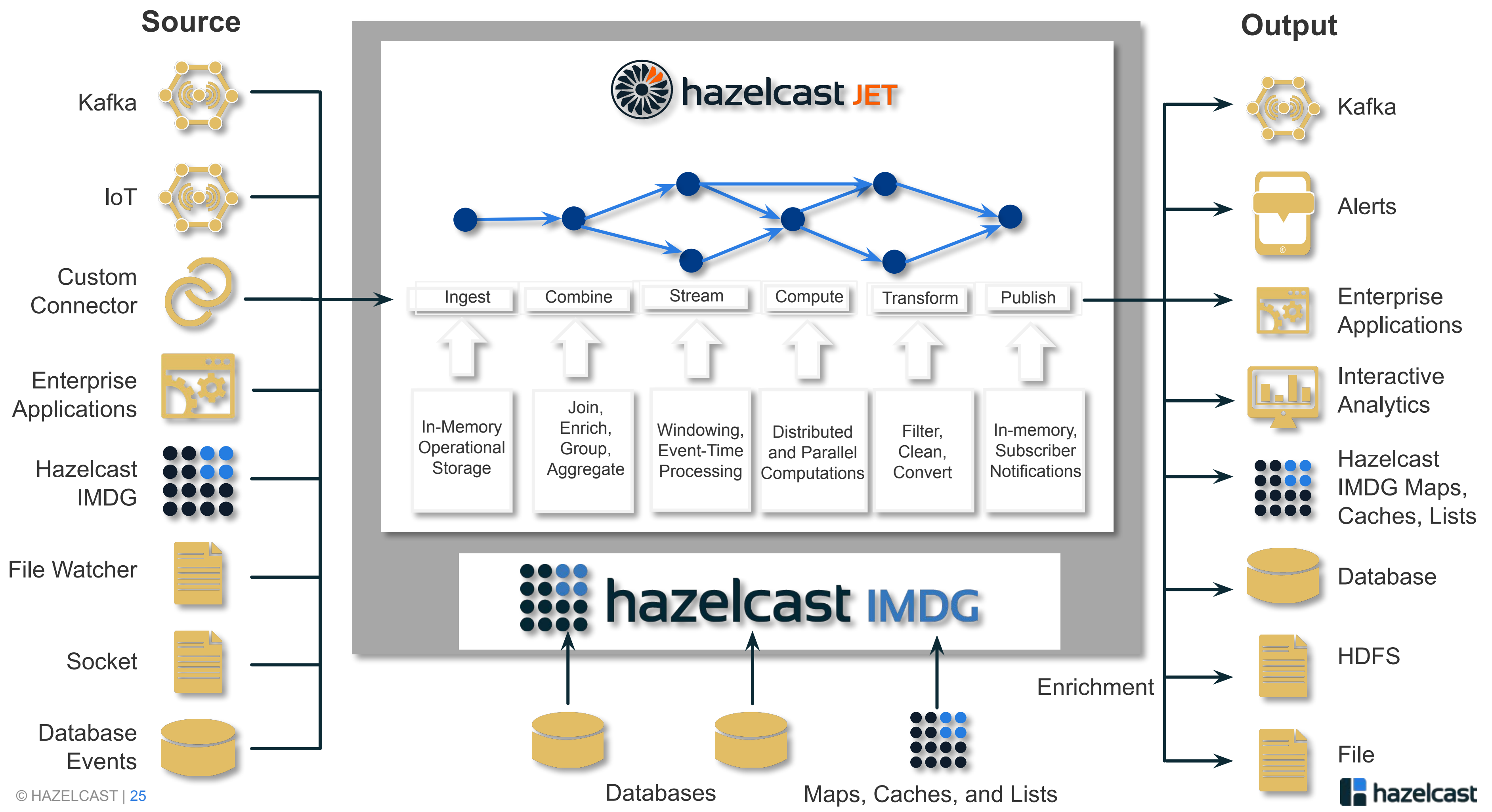
One-way keep DR site populated and ready-to-go

Two-way worldwide operations, access your nearest data copy

hazelcast

hazelcast

Hazelcast Jet Product Overview



Source
Kafka
IoT
Custom Connector
Enterprise Applications
Hazelcast IMDG
File Watcher
Socket
Database Events
hazelcast JET
Ingest
Combine
Stream
Compute
Transform
Publish
In-Memory Operational Storage
Join, Enrich, Group, Aggregate
Windowing, Event-Time Processing
Distributed and Parallel Computations
Filter, Clean, Convert
In-memory, Subscriber Notifications
hazelcast IMDG
Databases
Maps, Caches, and Lists
Enrichment
Output
Kafka
Alerts
Enterprise Applications
Interactive Analytics
Hazelcast IMDG Maps, Caches, Lists
Database
HDFS
File

hazelcast

Hazelcast Cloud
(Managed Service)

Hazelcast Cloud

Sign up here - https://hazelcast.com/products/cloud/

Your contributions are welcome!

- https://github.com/hazelcast/hazelcast
- https://github.com/hazelcast/hazelcast-jet

Resources

Downloads:

- https://hazelcast.org/
- https://hazelcast.org/imdg/
- https://jet-start.sh/

Demos

- https://github.com/hazelcast/hazelcast-jet-demos/tree/master/flight-telemetry
- **Cloud demo**

Free Training:

https://training.hazelcast.com

Join the Discussion:

Google Group: https://groups.google.com/forum/#!forum/hazelcast
Stack Overflow: http://stackoverflow.com/questions/tagged/hazelcast
Gitter: https://gitter.im/hazelcast/hazelcast

Thank You

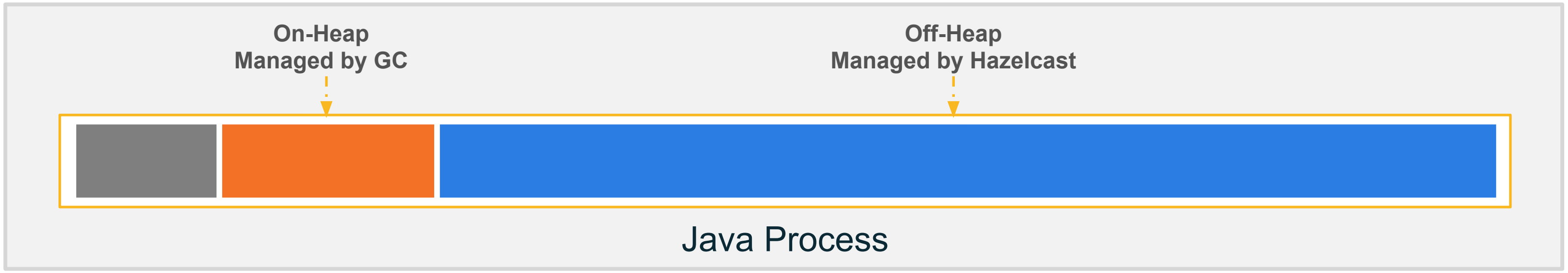
hazelcast

hazelcast

High-Density Memory Store

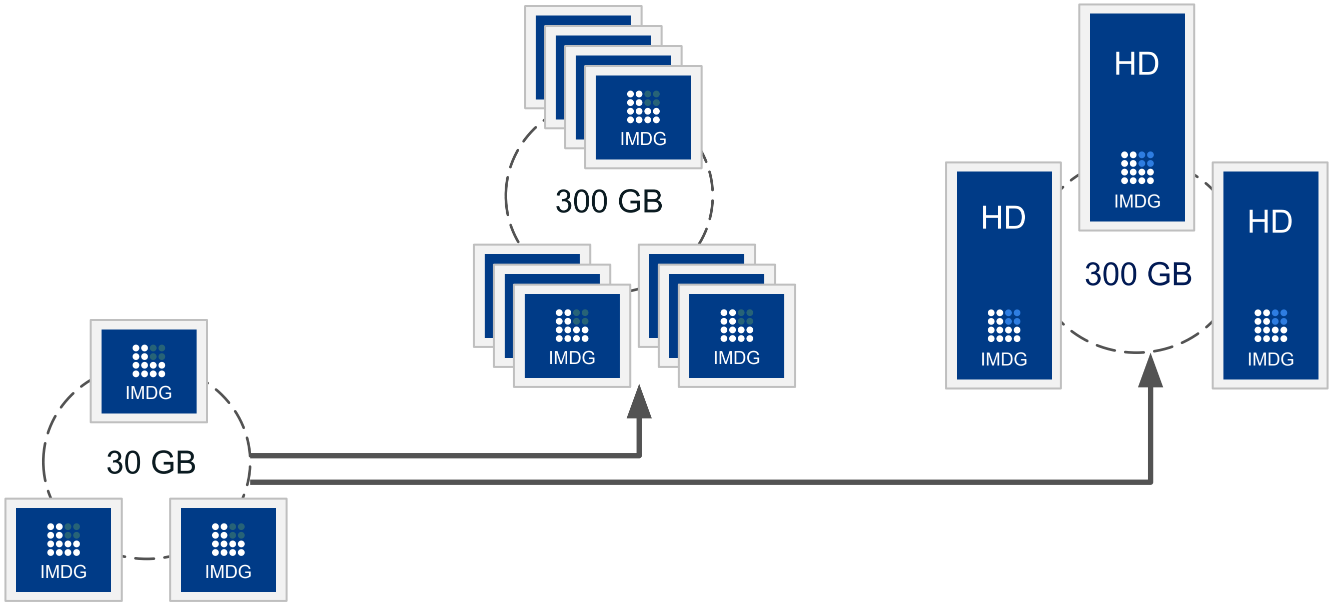
EE

High-Density Memory Store



- Maximizes use of available machine memory
- Eliminates exhausting black art of GC-tuning and ineffective workarounds
- Delivers improved performance and predictability for SLAs
- Support for Intel Optane as fast, lower cost, higher density volatile memory

High-Density Memory Store: Scale UP

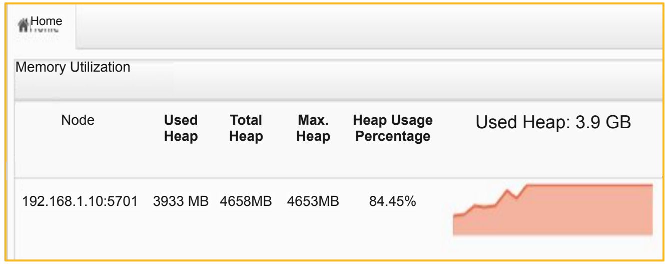


hazelcast

EE

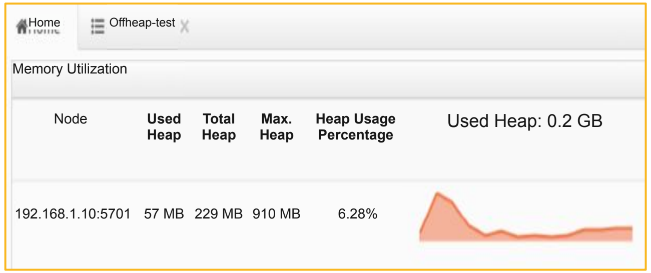
High-Density Memory Store: Before and After

On-Heap Memory		With HD Memory Store
0 MB	**Native Storage**	3.3 GB
3.9 GB	**Heap Storage**	0.6 GB
9 (4900 ms)	**Major GC**	0 (0 ms)
31 (4200 ms)	**Minor GC**	356 (349 ms)



Memory Utilization

Node	Used Heap	Total Heap	Max. Heap	Heap Usage Percentage	Used Heap: 3.9 GB
192.168.1.10:5701	3933 MB	4658MB	4653MB	84.45%	



Memory Utilization

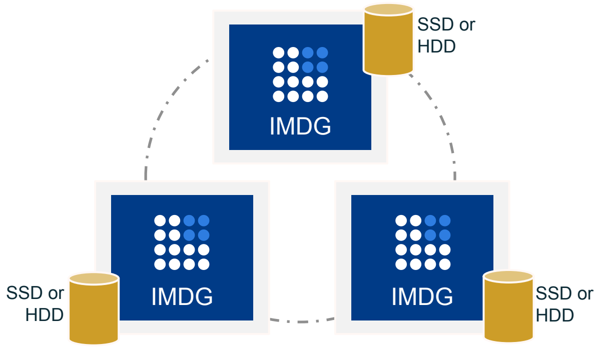
Node	Used Heap	Total Heap	Max. Heap	Heap Usage Percentage	Used Heap: 0.2 GB
192.168.1.10:5701	57 MB	229 MB	910 MB	6.28%	

hazelcast

Hot Restart Store

EE PS

Hot Restart Store
Restart your cluster without having to reload your data



Single Node		Reading		Writing
Storage	Data Size	Restart Time	Read Throughput	Write Throughput
SSD	100 GB	82 sec	1.4 GB/s	425,000 ops/sec
SSD	50 GB	49 sec	1.2 GB/s	400,000 ops/sec
SSD	25 GB	39 sec	0.8 GB/s	502,254 ops/sec
Intel Optane	87 GB	18 sec	4.83 GB/s	-

- Solves the most important operational problem with huge caches – need to reload data if you restart the cluster
- Purpose-designed, ultra high-performance persistence store, optimized for SSD
- Using Intel Optane DC Persistent Memory instead of SSDs enables significantly faster restart times

intel OPTANE DC PERSISTENT MEMORY

- Each node has its own store. Scales linearly across the cluster
- Data entirely loaded into RAM on start-up so Hazelcast IMDG always operates at in-memory speed
- Persistence configurable per data structure for JCache, Map, Web Sessions and Hibernate
- Partial recovery supported where some nodes are lost before restart
- Snapshotting for taking backups

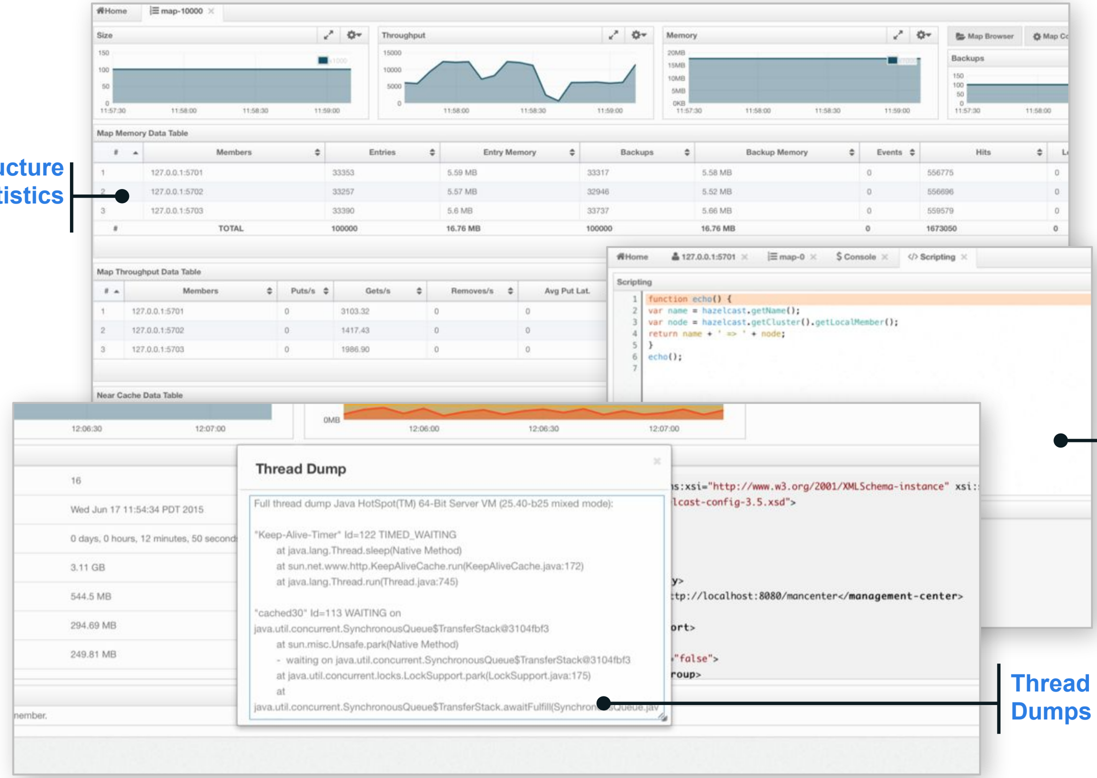
hazelcast

hazelcast

Management Center

EE PS

Management Center



Map Memory Data Table

#	Members	Entries	Entry Memory	Backups	Backup Memory	Events	Hits	L
1	127.0.0.1:5701	33353	5.59 MB	33317	5.58 MB	0	556775	0
2	127.0.0.1:5702	33257	5.57 MB	32946	5.52 MB	0	556696	0
3	127.0.0.1:5703	33390	5.6 MB	33737	5.66 MB	0	559579	0
#	TOTAL	100000	16.76 MB	100000	16.76 MB	0	1673050	0

Map Throughput Data Table

#	Members	Puts/s	Gets/s	Removes/s	Avg Put Lat.
1	127.0.0.1:5701	0	3103.32	0	0
2	127.0.0.1:5702	0	1417.43	0	0
3	127.0.0.1:5703	0	1986.90	0	0

Scripting

```
function echo() {
var name = hazelcast.getName();
var node = hazelcast.getCluster().getLocalMember();
return name + ' => ' + node;
}
echo();
```

16
Wed Jun 17 11:54:34 PDT 2015
0 days, 0 hours, 12 minutes, 50 second
3.11 GB
544.5 MB
294.69 MB
249.81 MB

Thread Dump

```
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.40-b25 mixed mode):

"Keep-Alive-Timer" Id=122 TIMED_WAITING
    at java.lang.Thread.sleep(Native Method)
    at sun.net.www.http.KeepAliveCache.run(KeepAliveCache.java:172)
    at java.lang.Thread.run(Thread.java:745)

"cached30" Id=113 WAITING on
java.util.concurrent.SynchronousQueue$TransferStack@3104fbf3
    at sun.misc.Unsafe.park(Native Method)
    -  waiting on java.util.concurrent.SynchronousQueue$TransferStack@3104fbf3
    at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
    at
java.util.concurrent.SynchronousQueue$TransferStack.awaitFulfill(Synchron  sQueue.jav
```

```
s:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
lcast-config-3.5.xsd">
y>
tp://localhost:8080/mancenter</management-center>
ort>
"false">
roup>
```

Other products integrated via Management Center APIs:

- MBean Tool
- JConsole
- VisualVM
- Hawtio

hazelcast

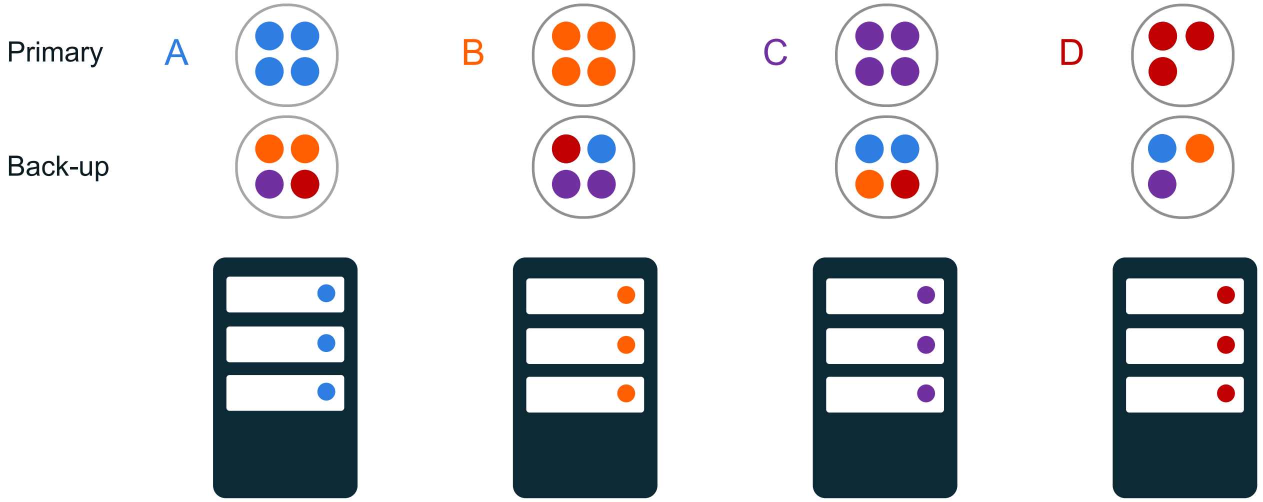
hazelcast

Data Rebalancing: How Back-ups Replicate across Nodes

Data Rebalancing: Migration Complete

Fixed number of partitions (default 271)

Each key falls into partition



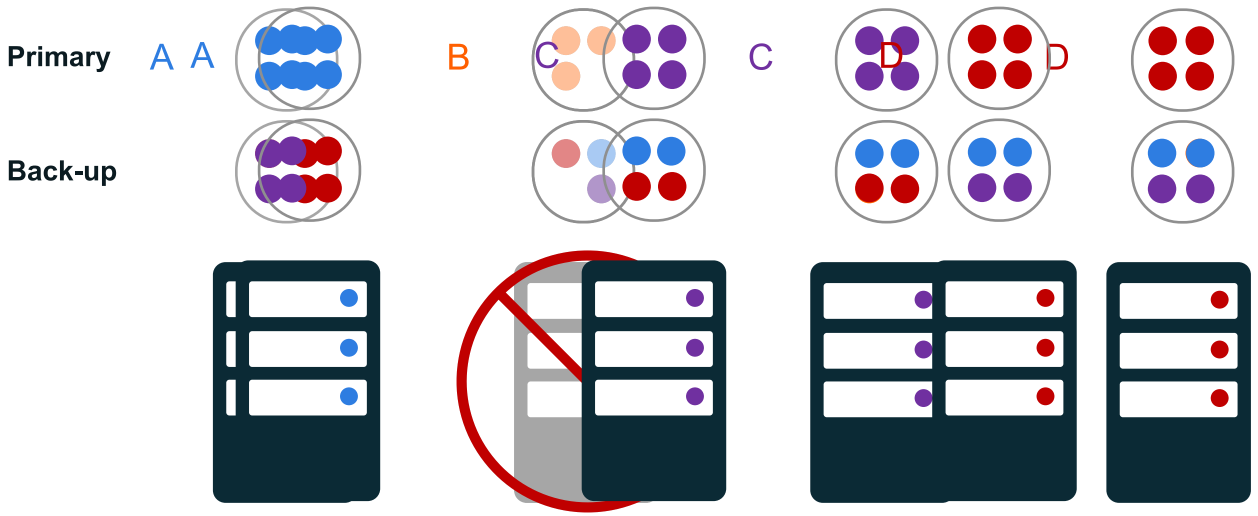
Partitionid = hash(keyData)%PARTITION_COUNT

Partition ownerships are reassigned upon membership change

hazelcast

Back-Up Recovery: Data Safety When Nodes Die

Back-up Recovery: Recovery Is Complete



APPENDIX

Analyst Reports

Hazelcast IMDG reviewed in Gartner "Market Guide for In-Memory Data Grids" [subscription required] https://www.gartner.com/doc/3420617/market-guide-inmemory-data-grids

"On the Radar: An open-source in-memory data grid platform for Java" [subscription required] https://www.ovumkc.com/Products/IT/Infrastructure-Solutions/On-the-Radar-Hazelcast/Summary

Hazelcast Inc. cited as Leader by Independent Research Firm
Forrester Wave In-Memory Data Grids September 2015