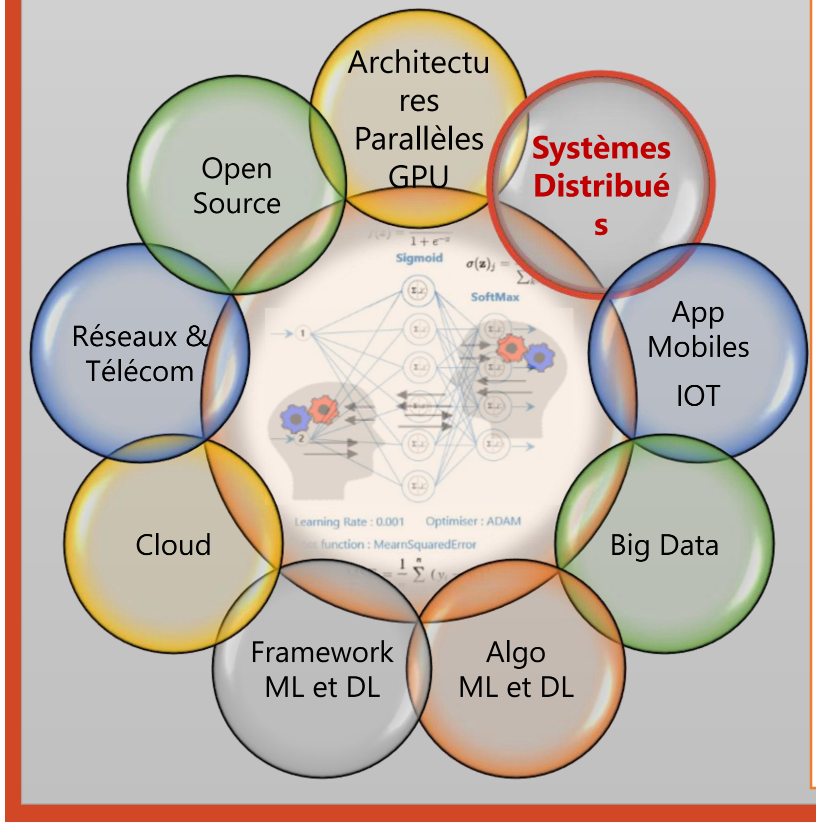
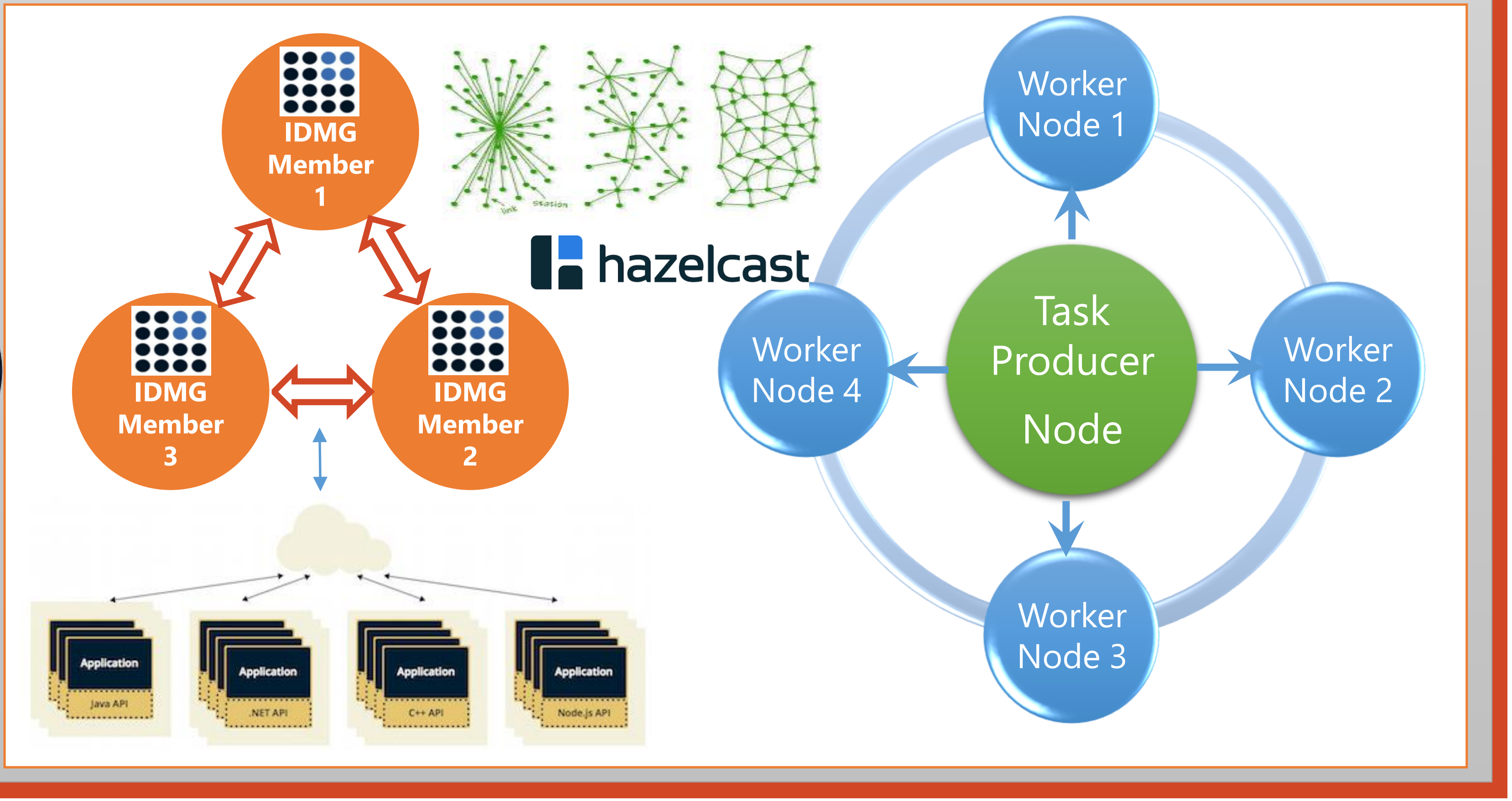
Distributed Computing and Caching with Hazelcast EcoSystem

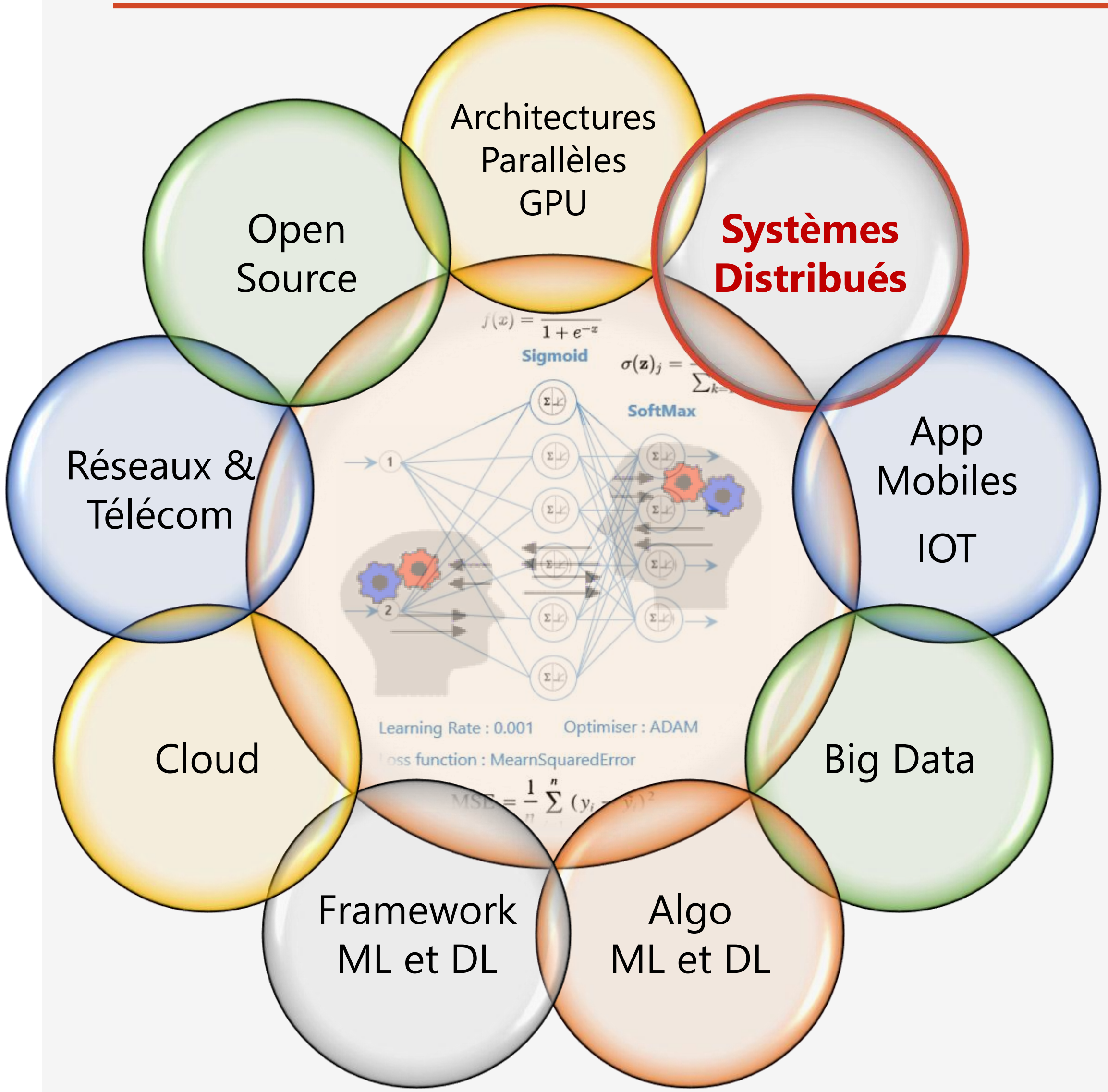
IDMG : In Distributed Memory Grid

By : Mohamed YOUSSFI
Lab. SSDIA, ENSET Mohammedia,
Hassan II University of Casablanca
Morocco
med@youssfi.net





Systèmes Distribués



Architectures Parallèles GPU

Systèmes Distribués

App Mobiles IOT

Big Data

Algo ML et DL

Framework ML et DL

Cloud

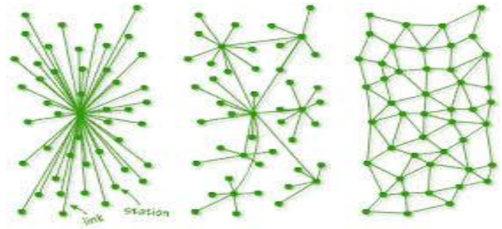
Réseaux & Télécom

Open Source

➢ Systèmes Distribués

- Middlewares RMI, CORBA, JMS, MPI
- Grilles de Clacul
- Protocoles de messageries asynchrones:
- **AMQP, MQTT, STOMP**
- Brokers : **RabbitMQ, ActiveMQ**
- Caches mémoires Distribués : **Hazelcast**
- Middlewares **SMA : JADE**

Systèmes Distribués Middlewares



CORBA

Java RMI

MPI

RabbitMQ

Apache ACTIVEMQ

AMQP

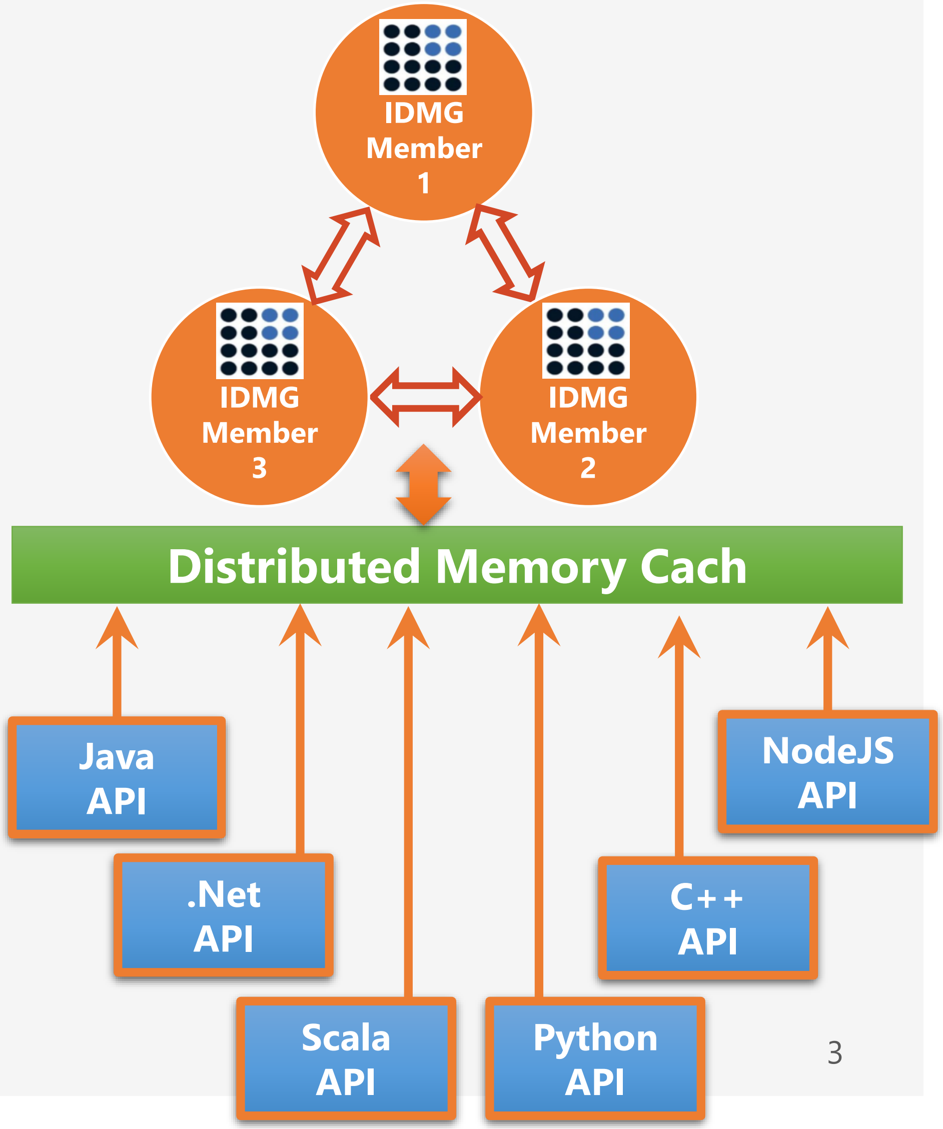
MQTT.ORG

Stomp

hazelcast

C'est quoi Hazelcast ?

- Hazelcast IMDG est un middleware Open source en Java, qui permet de créer un cache mémoire distribué.
- Dans une grille Hazelcast, les données sont réparties uniformément entre les nœuds d'un groupe d'ordinateurs, ce qui permet
  - Un stockage distribué Scalable (Distributed Memory Cache)
  - Un traitement distribué Scalable (Distributed Computing).
  - Réplication des données sur plusieurs nœud pour tolérance aux panes
- Ces techniques réduisent la charge de requête sur les bases de données et améliorent les performances des systèmes distribués.



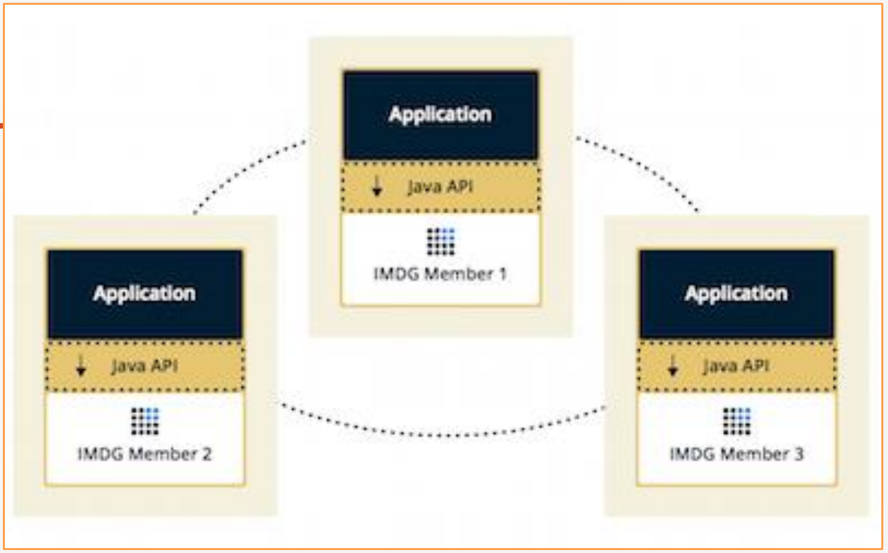
Utilisations de Hazelcast

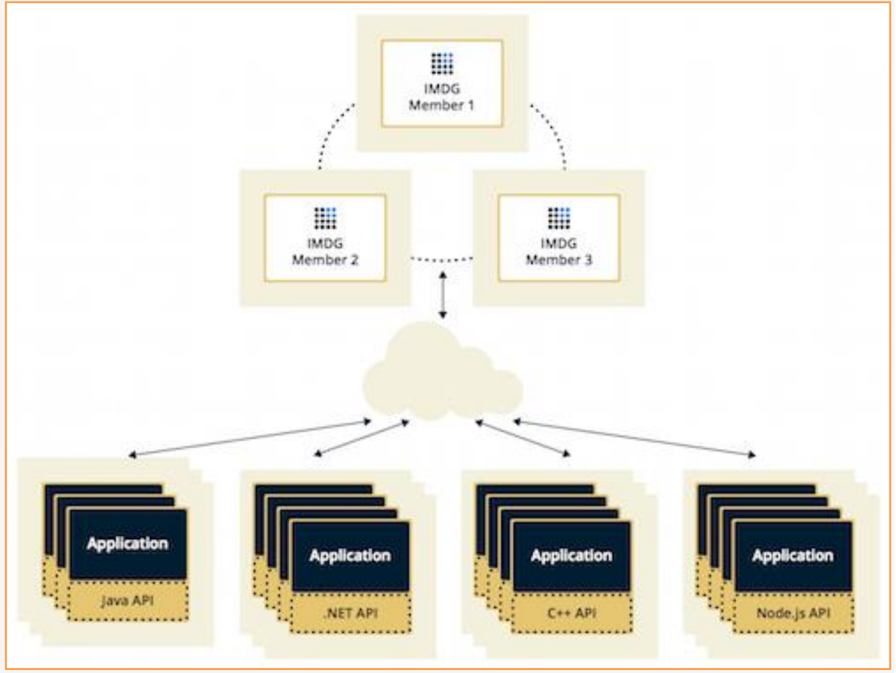
Hazelcast est une solution quand vous avez besoin de:

- Applications analytiques nécessitant un traitement de données volumineuses en partitionnant les données (Big Data)
- Pour conserver les données fréquemment consultées dans la grille mémoires
- Un cache mémoire distribué hautement disponible pour les applications
- Un magasin de données principal pour les applications avec des exigences de performances, d'évolutivité et de latence maximales
- In Memory NoSQL data base de type Clé-Valeurs
- Solution de messagerie (publier / souscrire) à très rapide avec un scalabilité entre les applications
- Solution faire distribuer les traitements (Distributed Computing)
- Une alternative aux autres solutions comme Coherence and Terracotta.

C'est quoi Hazelcast ?

- Hazelcast est implémenté en Java et possède des clients pour Java, C / C ++, .NET, REST, Python, Go et Node.js.
- Contrairement à beaucoup de solutions NoSQL, Hazelcast est peer-to-peer. Il n'y a pas de maître et d'esclave;
- il n'y a pas point de défaillance unique (SPOF).
- Système d'équilibrage de charges : Tous les membres stockent des quantités égales de données et font des quantités égales de En traitement.
- On peut intégrer Hazelcast dans une application existante
- ou l'utiliser pour faire de votre application est un client pour un cluster Hazelcast.





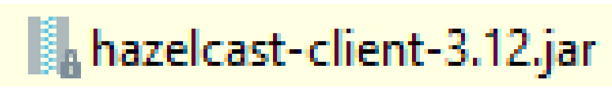
Caractéristiques de Hazelcast

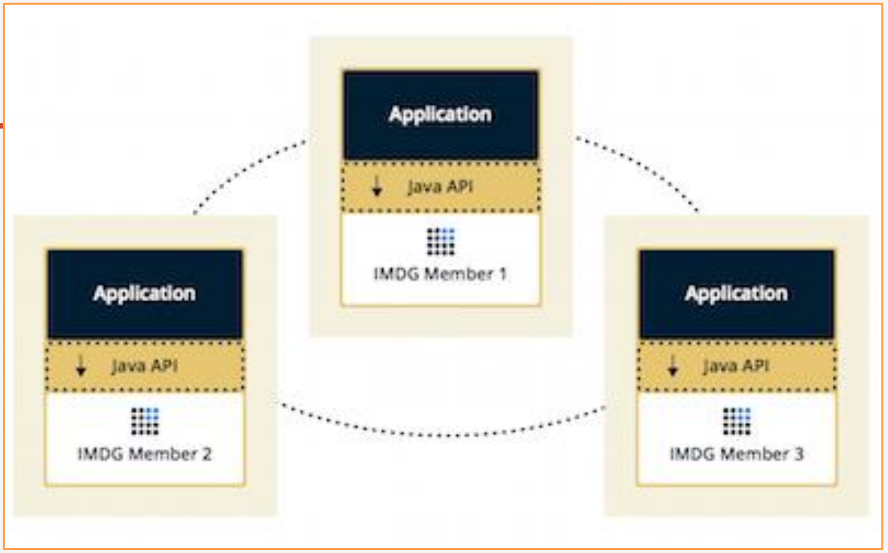
- Hazelcast est une solution Opensource
- Pour utilizer Hazelcast vous avez besoin uniquement d’un seul fichier JAR.
- Hazelcast fournit une implementation distribuée des structures de données classiques acomme
  - Map, Queue, MultiMap, Topic, Lock Executor.
- On peut ajouter de Nouvelles implementation de structures de données distribuées en utilisant le Service Programming Interface (SPI)
- L’architecture de Hazecast est comlètement distribuée :
  - Il n’y a pas de noeud Master centralisé
  - Pas de point d’échec (SPOF)
- Tous les noeuds sont configurés pour être identiques
- Quand les capacités mémoire et de calcul nécessitent de croitre, Il suffit de démare de nouveaux membres faisant partie du Cluster (Scalability)
- Les données sont résilientes à l’échec d’un membre , vu que des backups des données sont duppliquées dans plsusieurs noeuds du cluster

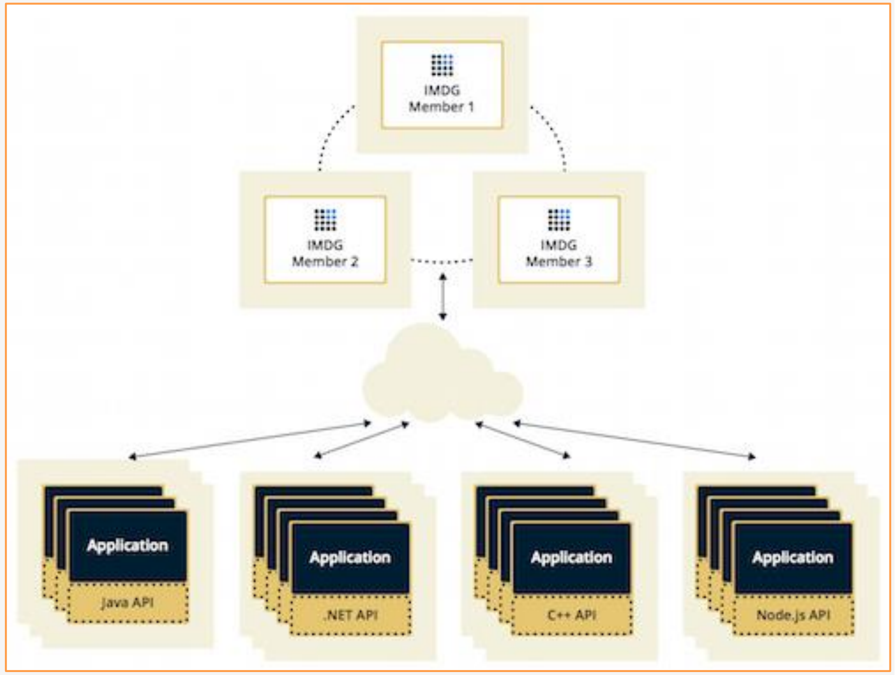
Intégration de Hazel Cast

```
<dependencies>
    <dependency>
        <groupId>com.hazelcast</groupId>
        <artifactId>hazelcast</artifactId>
        <version>3.12</version>
    </dependency>
    <dependency>
        <groupId>com.hazelcast</groupId>
        <artifactId>hazelcast-client</artifactId>
        <version>3.12</version>
    </dependency>
</dependencies>
```

hazelcast-3.12.jar


hazelcast-client-3.12.jar





Première Application Hazelcast

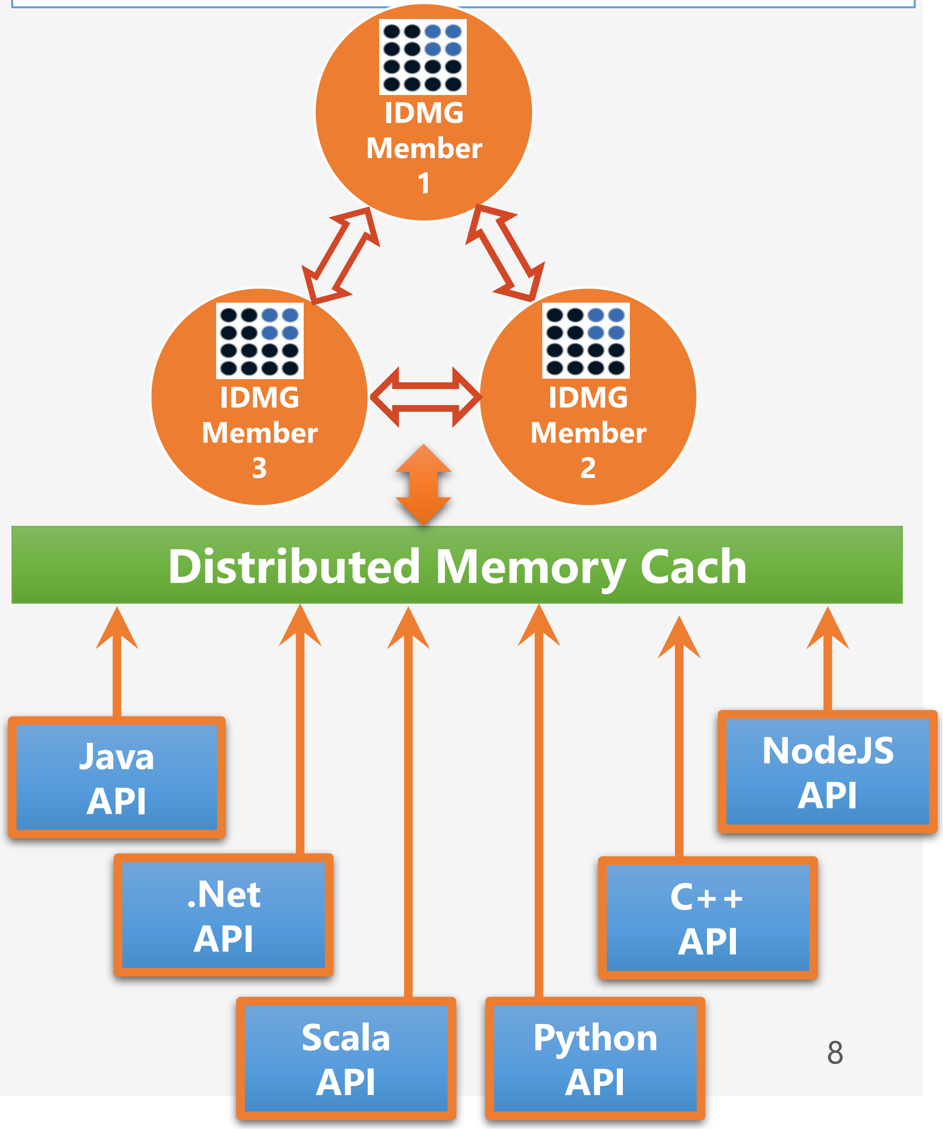
- En exécutant cette application en plusieurs instances, vous créez une grille mémoire distribuée
- Les données des deux collections Map et Queue sont réparties sur plusieurs nœuds.
- Les deux collections distribuées sont partagées à toutes les instances.

Démarrage d'un nœud du cluster hazelcast

```java
Config cfg = new Config();
HazelcastInstance instance = Hazelcast.newHazelcastInstance(cfg);
Map<Integer, String> mapCustomers = instance.getMap("customers");
mapCustomers.put(1, "Joe");
mapCustomers.put(2, "Ali");
mapCustomers.put(3, "Avi");
System.out.println("Customer with key 1: "+ mapCustomers.get(1));
System.out.println("Map Size:" + mapCustomers.size());
Queue<String> queueCustomers = instance.getQueue("customers");
queueCustomers.offer("Tom");
queueCustomers.offer("Mary");
queueCustomers.offer("Jane");
System.out.println("First customer: " + queueCustomers.poll());
System.out.println("Second customer: "+ queueCustomers.peek());
System.out.println("Queue size: " + queueCustomers.size());
```

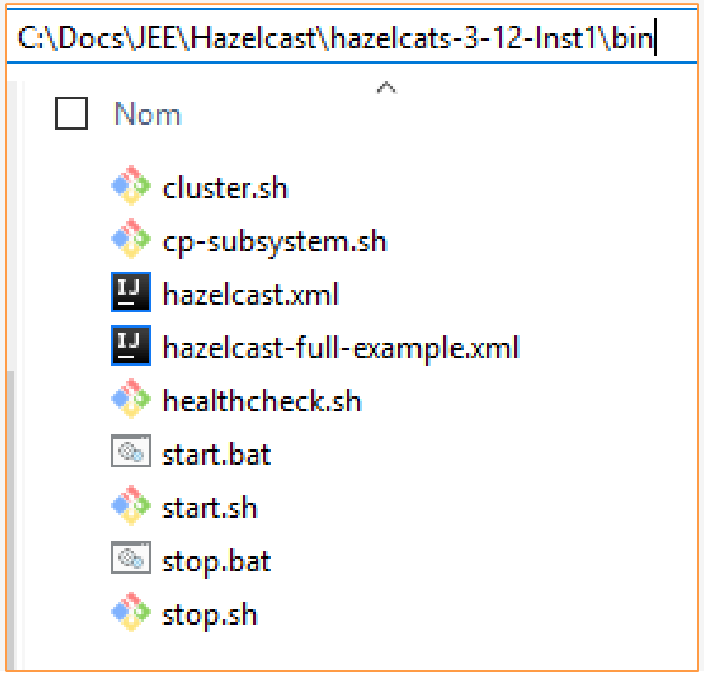
Démarrage d'un client du cluster hazelcast

```java
ClientConfig clientConfig = new ClientConfig();
HazelcastInstance client = HazelcastClient.newHazelcastClient(
clientConfig);
IMap map = client.getMap( "customers" );
System.out.println( "Map Size:" + map.size() );
```



Démarrage des nœuds Hazelcast avec les scripts fournis

- Télécharger la distribution de Hazelcast DMG:
  - https://hazelcast.org/download/

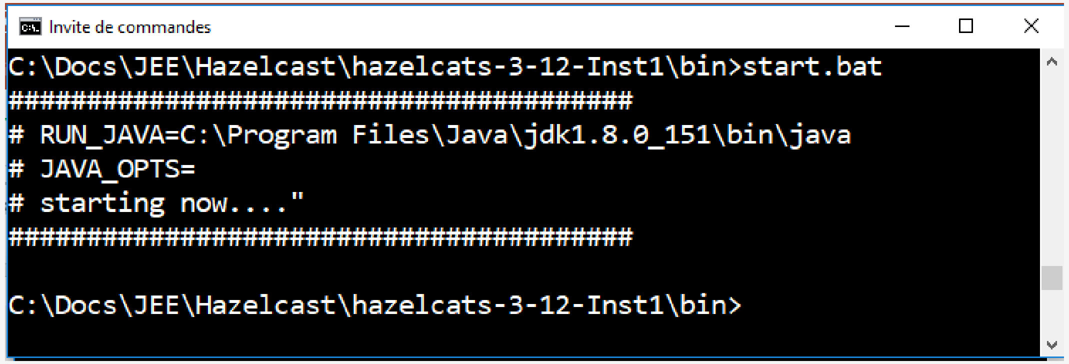


C:\Docs\JEE\Hazelcast\hazelcats-3-12-Inst1\bin

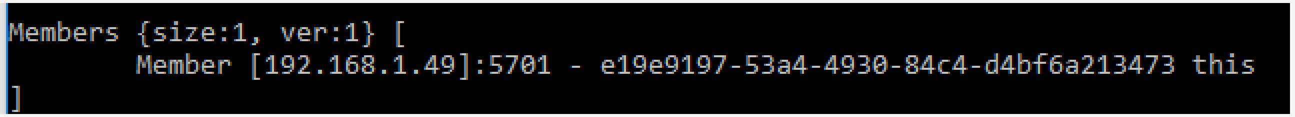
□ Nom

- cluster.sh
- cp-subsystem.sh
- hazelcast.xml
- hazelcast-full-example.xml
- healthcheck.sh
- start.bat
- start.sh
- stop.bat
- stop.sh

Démarrage d'un nœud Hazelcast

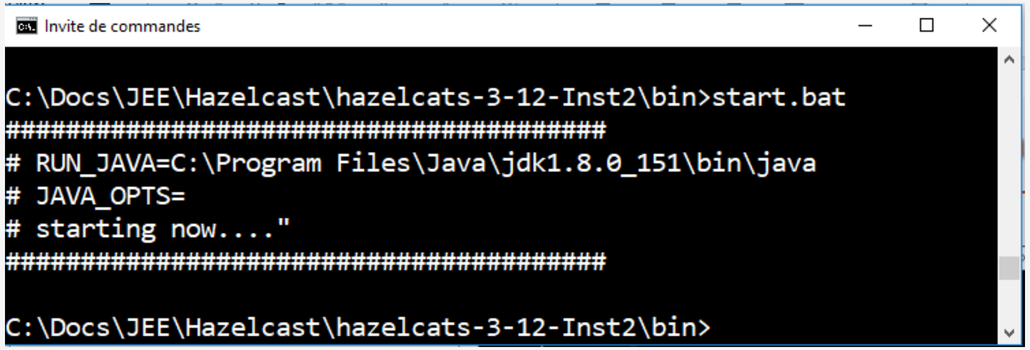


```
C:\Docs\JEE\Hazelcast\hazelcats-3-12-Inst1\bin>start.bat
########################################
# RUN_JAVA=C:\Program Files\Java\jdk1.8.0_151\bin\java
# JAVA_OPTS=
# starting now...."
########################################

C:\Docs\JEE\Hazelcast\hazelcats-3-12-Inst1\bin>
```

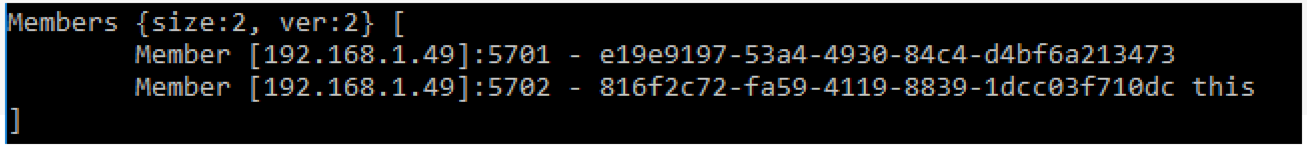


```
Members {size:1, ver:1} [
        Member [192.168.1.49]:5701 - e19e9197-53a4-4930-84c4-d4bf6a213473 this
]
```

Démarrage d'un autre nœud Hazelcast



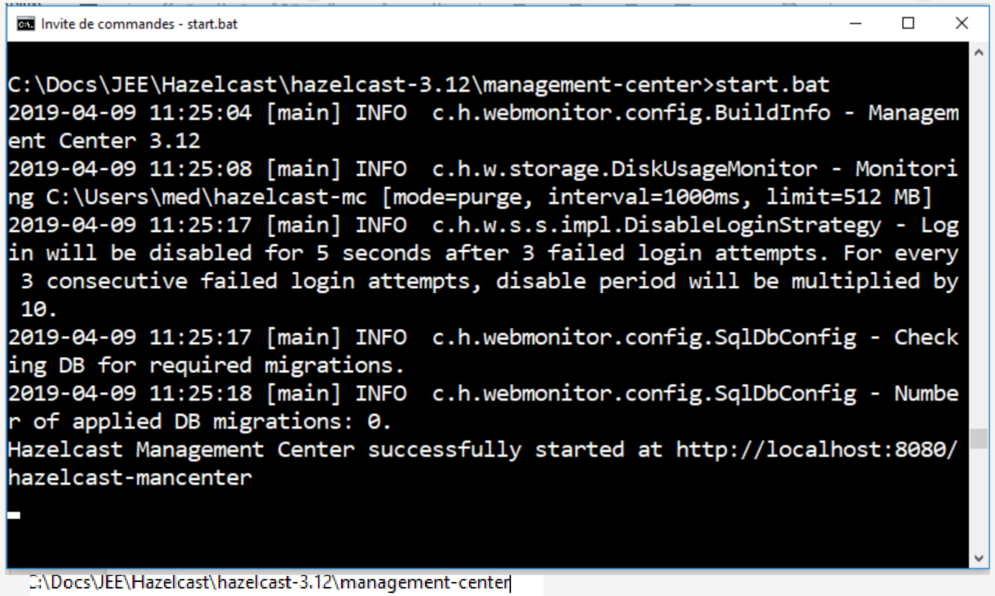
```
C:\Docs\JEE\Hazelcast\hazelcats-3-12-Inst2\bin>start.bat
########################################
# RUN_JAVA=C:\Program Files\Java\jdk1.8.0_151\bin\java
# JAVA_OPTS=
# starting now...."
########################################

C:\Docs\JEE\Hazelcast\hazelcats-3-12-Inst2\bin>
```



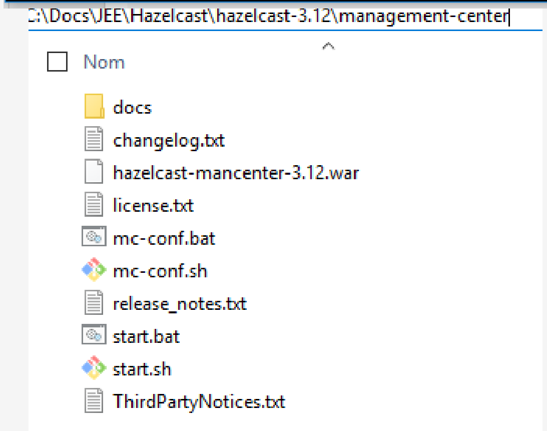
```
Members {size:2, ver:2} [
        Member [192.168.1.49]:5701 - e19e9197-53a4-4930-84c4-d4bf6a213473
        Member [192.168.1.49]:5702 - 816f2c72-fa59-4119-8839-1dcc03f710dc this
]
```

Démarrage de la console de de monitoring et de management Hazelcast

Démarrage de la console de management



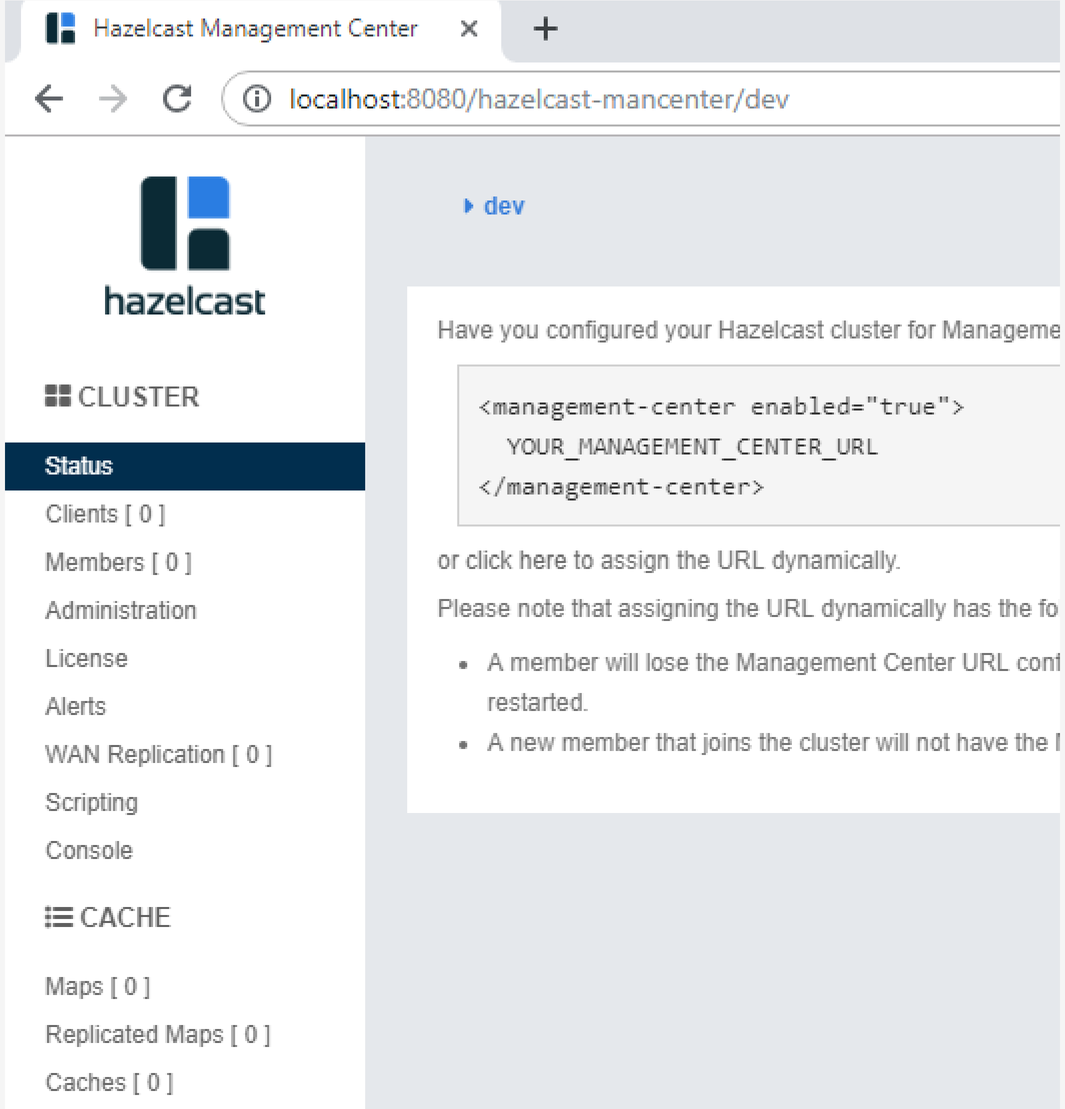
```
C:\Docs\JEE\Hazelcast\hazelcast-3.12\management-center>start.bat
2019-04-09 11:25:04 [main] INFO  c.h.webmonitor.config.BuildInfo - Managem
ent Center 3.12
2019-04-09 11:25:08 [main] INFO  c.h.w.storage.DiskUsageMonitor - Monitori
ng C:\Users\med\hazelcast-mc [mode=purge, interval=1000ms, limit=512 MB]
2019-04-09 11:25:17 [main] INFO  c.h.w.s.s.impl.DisableLoginStrategy - Log
in will be disabled for 5 seconds after 3 failed login attempts. For every
 3 consecutive failed login attempts, disable period will be multiplied by
 10.
2019-04-09 11:25:17 [main] INFO  c.h.webmonitor.config.SqlDbConfig - Check
ing DB for required migrations.
2019-04-09 11:25:18 [main] INFO  c.h.webmonitor.config.SqlDbConfig - Numbe
r of applied DB migrations: 0.
Hazelcast Management Center successfully started at http://localhost:8080/
hazelcast-mancenter
```



```
:\Docs\JEE\Hazelcast\hazelcast-3.12\management-center
```

□ Nom

- docs
- changelog.txt
- hazelcast-mancenter-3.12.war
- license.txt
- mc-conf.bat
- mc-conf.sh
- release_notes.txt
- start.bat
- start.sh
- ThirdPartyNotices.txt



Have you configured your Hazelcast cluster for Manageme

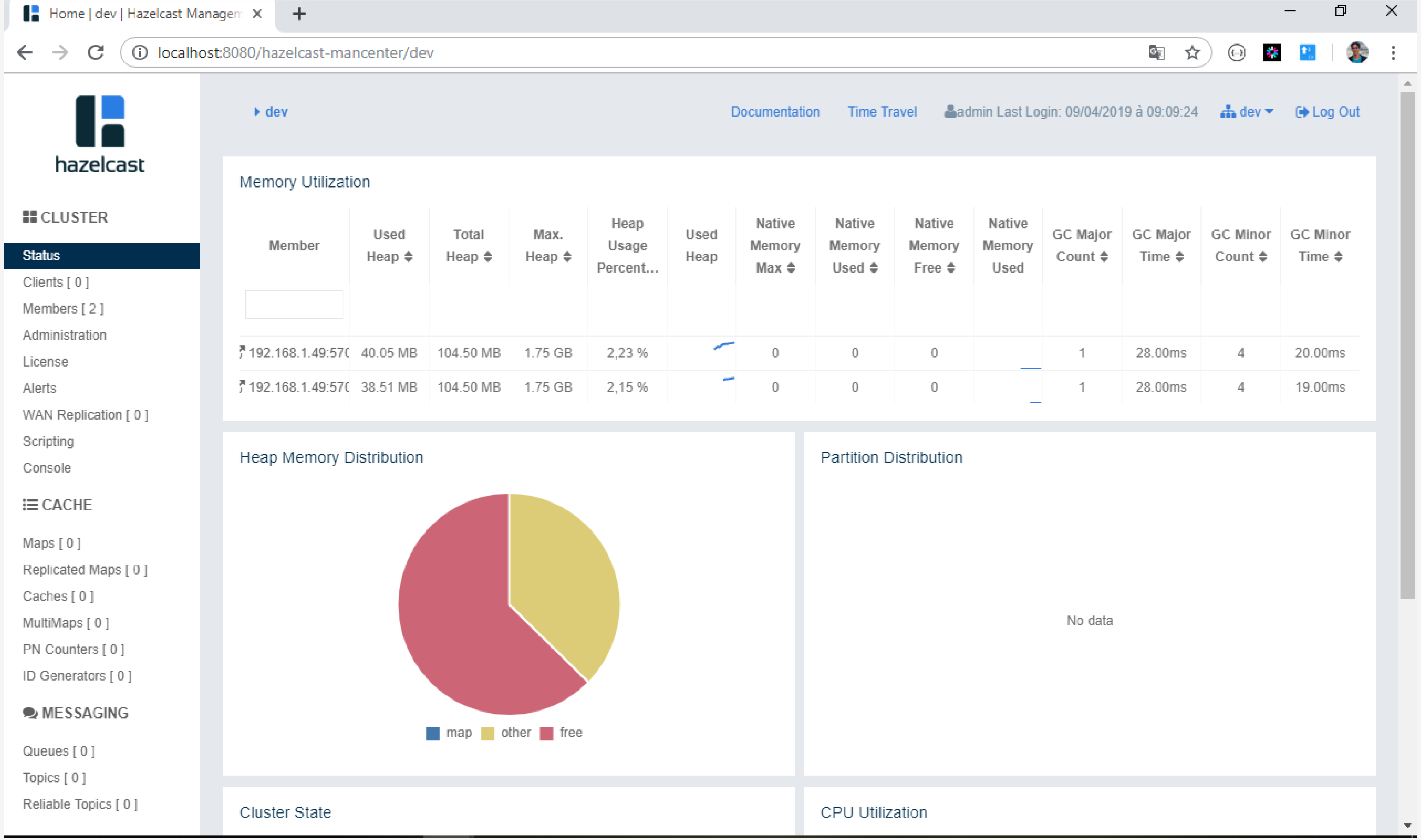
```
<management-center enabled="true">
  YOUR_MANAGEMENT_CENTER_URL
</management-center>
```

or click here to assign the URL dynamically.

Please note that assigning the URL dynamically has the fo

- A member will lose the Management Center URL conf restarted.
- A new member that joins the cluster will not have the l

Démarrage de la console de de monitoring et de management Hazelcast



Memory Utilization

Member	Used Heap	Total Heap	Max. Heap	Heap Usage Percent...	Used Heap	Native Memory Max	Native Memory Used	Native Memory Free	Native Memory Used	GC Major Count	GC Major Time	GC Minor Count	GC Minor Time
192.168.1.49:57(	40.05 MB	104.50 MB	1.75 GB	2,23 %		0	0	0		1	28.00ms	4	20.00ms
192.168.1.49:57(	38.51 MB	104.50 MB	1.75 GB	2,15 %		0	0	0		1	28.00ms	4	19.00ms

Heap Memory Distribution

Partition Distribution

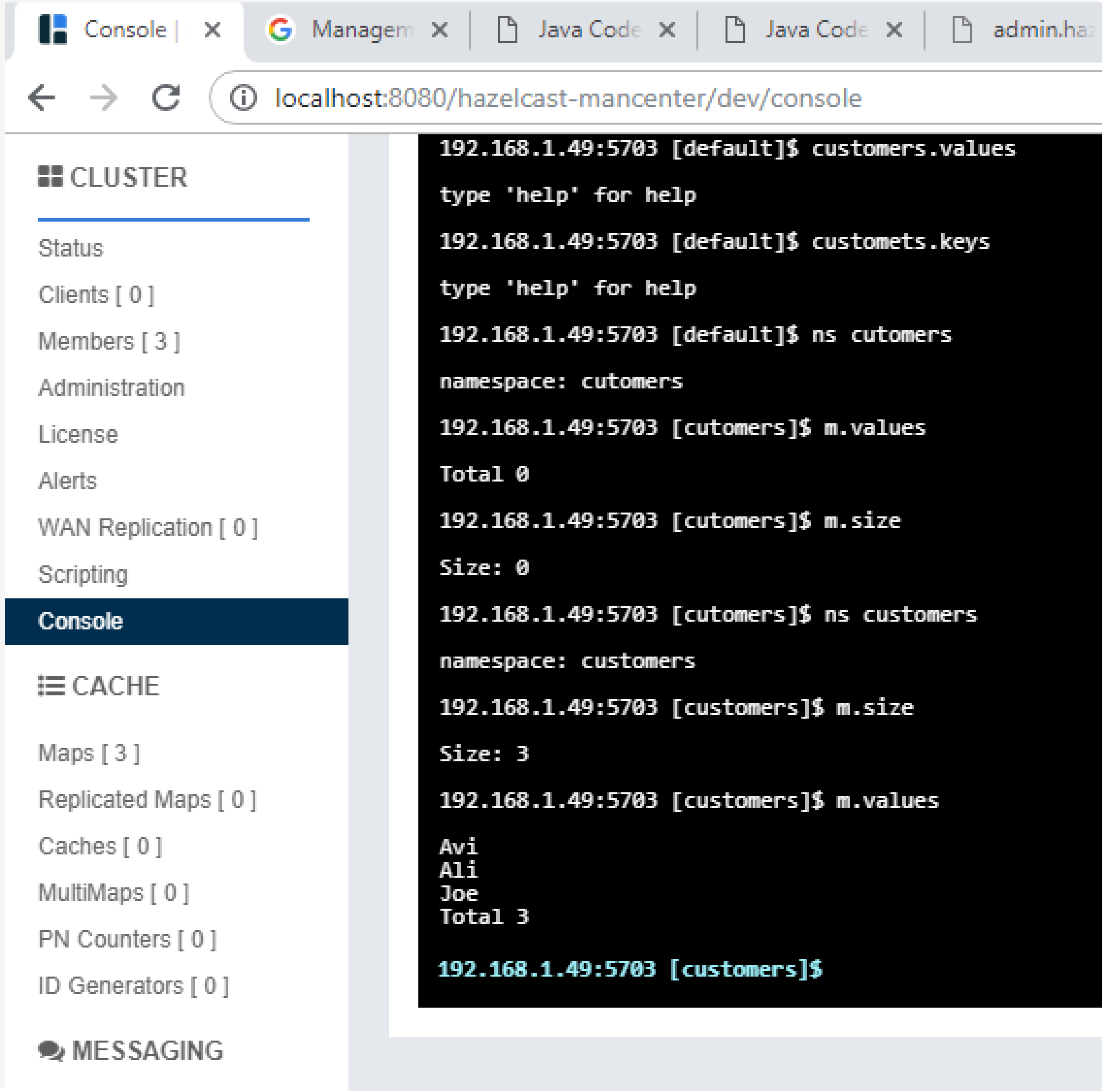
No data

Cluster State

CPU Utilization

Connecter un nœud hazelcast à hazelcast management center

```
ManagementCenterConfig mcfg =new
ManagementCenterConfig();

mcfg.setEnabled(true);

mcfg.setUrl("http://localhost:8080/hazelcast-mancenter");

Config cfg = new Config();

cfg.setManagementCenterConfig(mcfg);

HazelcastInstance instance =
Hazelcast.newHazelcastInstance(cfg);
```

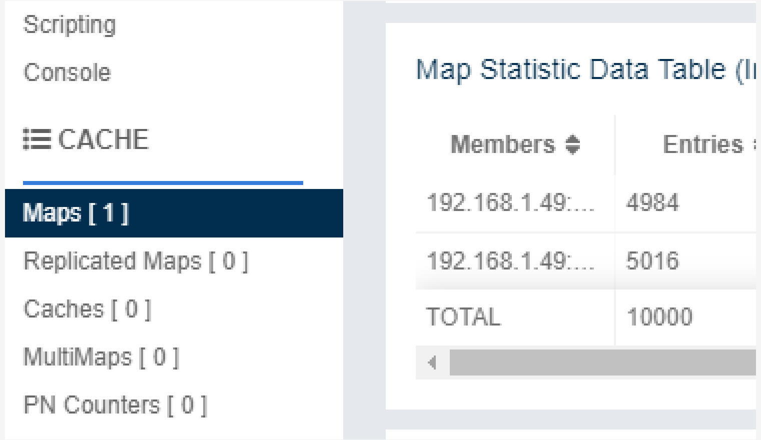


```
192.168.1.49:5703 [default]$ customers.values
type 'help' for help
192.168.1.49:5703 [default]$ customets.keys
type 'help' for help
192.168.1.49:5703 [default]$ ns cutomers
namespace: cutomers
192.168.1.49:5703 [cutomers]$ m.values
Total 0
192.168.1.49:5703 [cutomers]$ m.size
Size: 0
192.168.1.49:5703 [cutomers]$ ns customers
namespace: customers
192.168.1.49:5703 [customers]$ m.size
Size: 3
192.168.1.49:5703 [customers]$ m.values
Avi
Ali
Joe
Total 3

192.168.1.49:5703 [customers]$
```

Connecter un nœud hazelcast à hazelcast management center

```java
ManagementCenterConfig mcfg =new ManagementCenterConfig();
mcfg.setEnabled(true);
mcfg.setUrl("http://localhost:8080/hazelcast-mancenter");
Config cfg = new Config();
cfg.setManagementCenterConfig(mcfg);
HazelcastInstance instance = Hazelcast.newHazelcastInstance(cfg);
Map<Integer, String> mapCustomers = instance.getMap("customers");
for (int i = 0; i <10000 ; i++) {
    mapCustomers.put(i, "Data "+i);
}
```

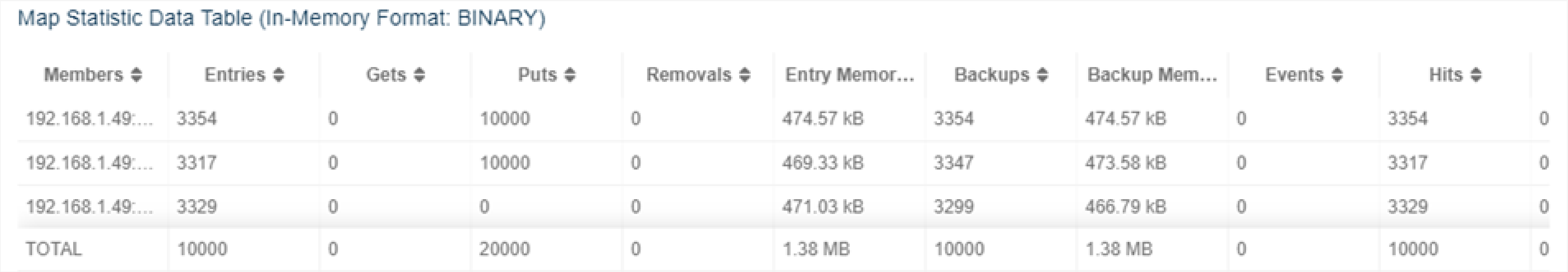
Après arrêt d'une instance



Map Statistic Data Table (I

Members ⬍	Entries
192.168.1.49:...	4984
192.168.1.49:...	5016
TOTAL	10000

3 instances

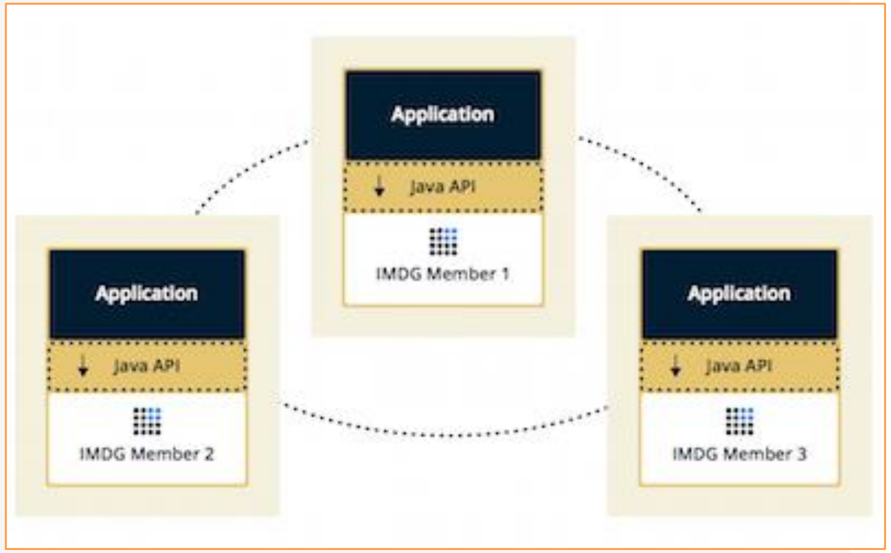


Map Statistic Data Table (In-Memory Format: BINARY)

Members ⬍	Entries ⬍	Gets ⬍	Puts ⬍	Removals ⬍	Entry Memor...	Backups ⬍	Backup Mem...	Events ⬍	Hits ⬍	
192.168.1.49:...	3354	0	10000	0	474.57 kB	3354	474.57 kB	0	3354	0
192.168.1.49:...	3317	0	10000	0	469.33 kB	3347	473.58 kB	0	3317	0
192.168.1.49:...	3329	0	0	0	471.03 kB	3299	466.79 kB	0	3329	0
TOTAL	10000	0	20000	0	1.38 MB	10000	1.38 MB	0	10000	0

Distributed Computing with Hazelcast : Callable Task

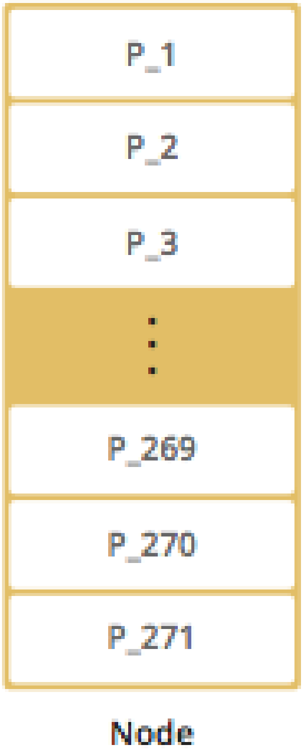
- Hazelcast offers IExecutorService for you to use in distributed environments.
- It implements java.util.concurrent.ExecutorService to serve the applications requiring computational and data processing power.
- With IExecutorService, you can execute tasks asynchronously and perform other useful tasks.
- If your task execution takes longer than expected, you can cancel the task execution.
- Tasks should be Serializable since they are distributed.
- In the Java Executor framework, you implement tasks two ways: Callable or Runnable.
  - Callable: If you need to return a value and submit it to Executor, implement the task as **java.util.concurrent.Callable**.
  - Runnable: If you do not need to return a value, implement the task as **java.util.concurrent.Runnable**.



Data Partition

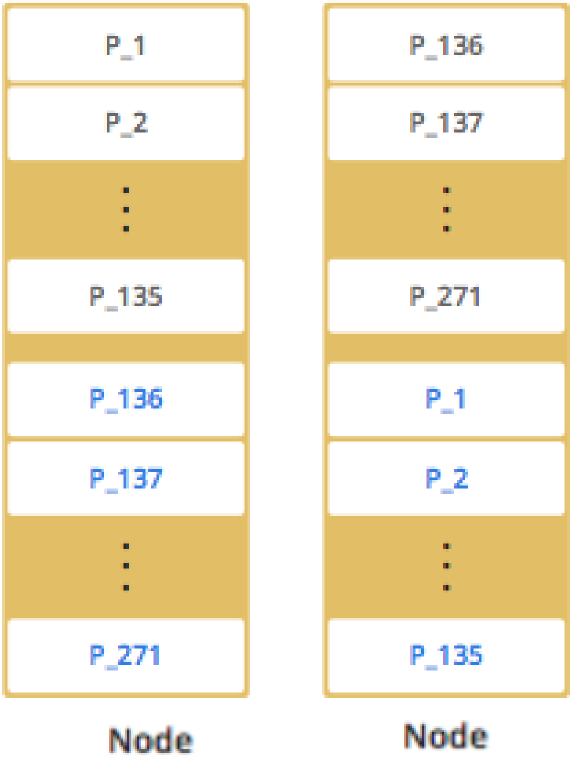
- Les fragments Hazelcast sont appelés des **partitions**.
- Les partitions sont des segments de mémoire pouvant contenir des centaines, voire des milliers, d'entrées de données, en fonction de la capacité de mémoire de votre système.
- Chaque partition Hazelcast peut avoir **plusieurs répliques (Copies)**, qui sont répartis entre les membres du cluster.
- Une des répliques devient la **réplique principale** et d’autres sont appelés des **sauvegardes**.
- Un membre du cluster qui possède le réplica principal d'une partition est appelé **partition propriétaire.**
- Lorsque vous lisez ou écrivez une entrée de données particulière, vous **parlez de manière transparente au propriétaire de partition** contenant les données.
- Par défaut, Hazelcast propose **271 partitions.**
- Lorsque vous démarrez un cluster avec un seul membre, il possède toutes les 271 partitions.
- En démarrant d’autres membres **les partitions sont distribuées équitablement aux membres du cluster**

Avec un membre



Avec deux membre

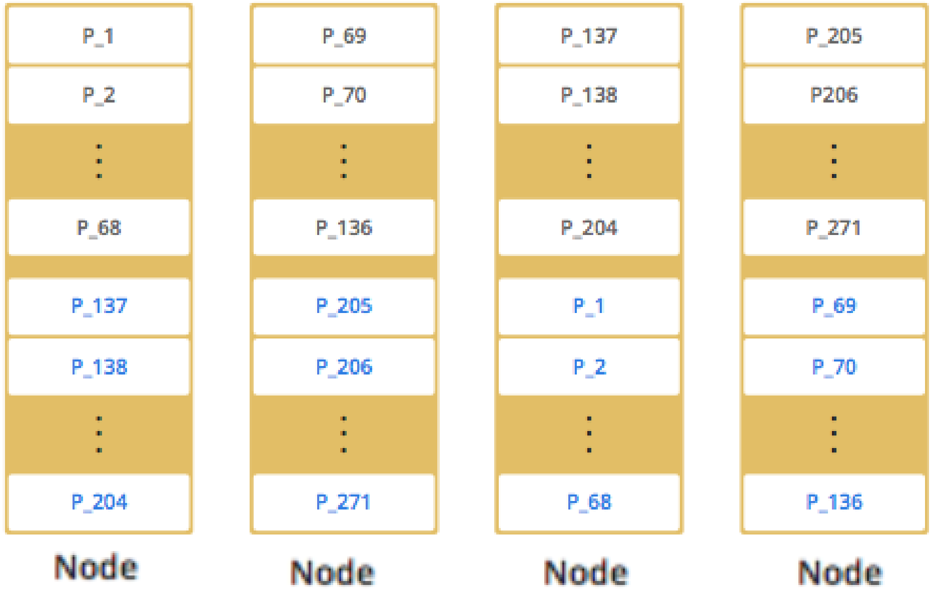
- En noir, les partions primaires
- En bleu, les partitions backup



Data Partition

- Les fragments Hazelcast sont appelés des partitions.
- Les partitions sont des segments de mémoire pouvant contenir des centaines, voire des milliers, d'entrées de données, en fonction de la capacité de mémoire de votre système.
- Chaque partition Hazelcast peut avoir plusieurs répliques (Copies), qui sont répartis entre les membres du cluster.
- Une des répliques devient la réplique principale et d’autres sont appelés des sauvegardes.
- Un membre du cluster qui possède le réplica principal d'une partition est appelé partition propriétaire.
- Lorsque vous lisez ou écrivez une entrée de données particulière, vous parlez de manière transparente au propriétaire de partition contenant les données.
- Par défaut, Hazelcast propose 271 partitions.
- Lorsque vous démarrez un cluster avec un seul membre, il possède toutes les 271 partitions.
- En démarrant d’autres membres les partitions sont distribuées équitablement aux membres du cluster

Pour un cluster de 4 membre

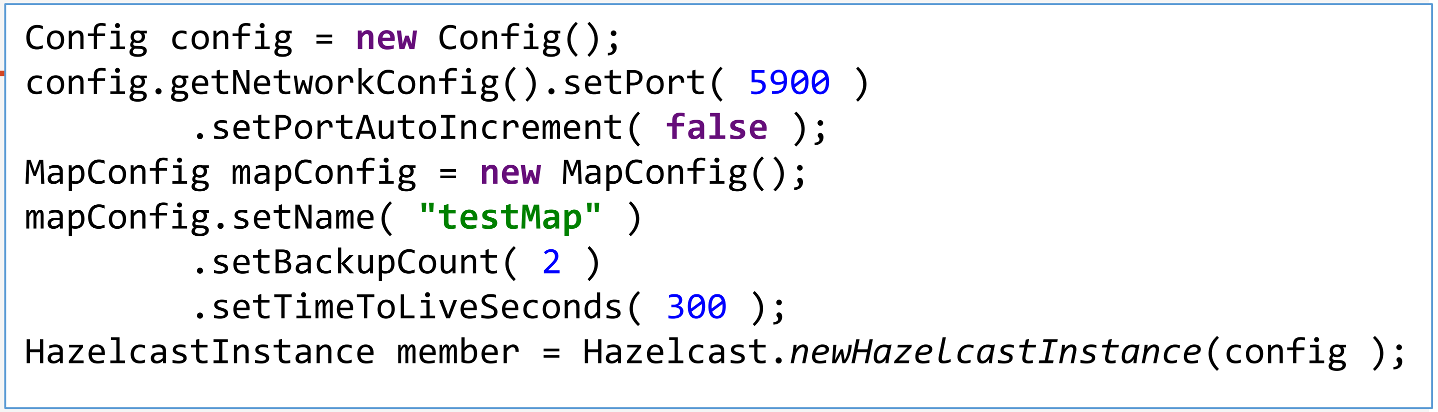


- Hazelcast distribue les partitions principale et secondaire (backup) de manière égale entre les membres du grappe. Les répliques de sauvegarde des partitions sont conservées pour la redondance.

Hazelcast Configuration

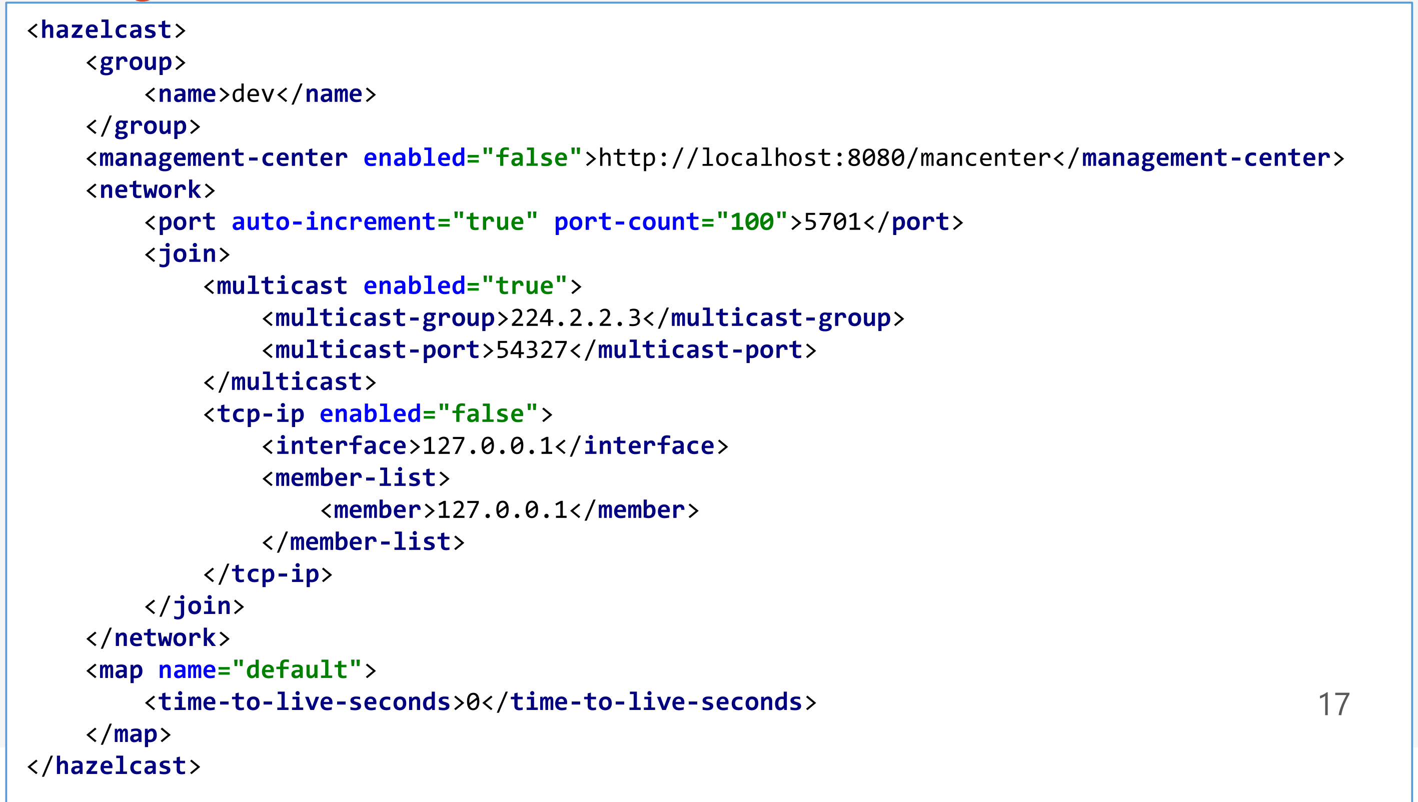
- Vous pouvez configurer Hazelcast en utilisant un ou plusieurs combinaisons des options suivantes:
  - manière déclarative (XML or a YAML File)
  - manière programmatique (Code Java ou autre)
  - Utilisation des propriétés du système Hazelcast
  - Dans le contexte du Framework Spring
  - Ajout dynamique de la configuration sur un cluster en cours d'exécution

Configuration programmatique



```java
Config config = new Config();
config.getNetworkConfig().setPort( 5900 )
        .setPortAutoIncrement( false );
MapConfig mapConfig = new MapConfig();
mapConfig.setName( "testMap" )
        .setBackupCount( 2 )
        .setTimeToLiveSeconds( 300 );
HazelcastInstance member = Hazelcast.newHazelcastInstance(config );
```

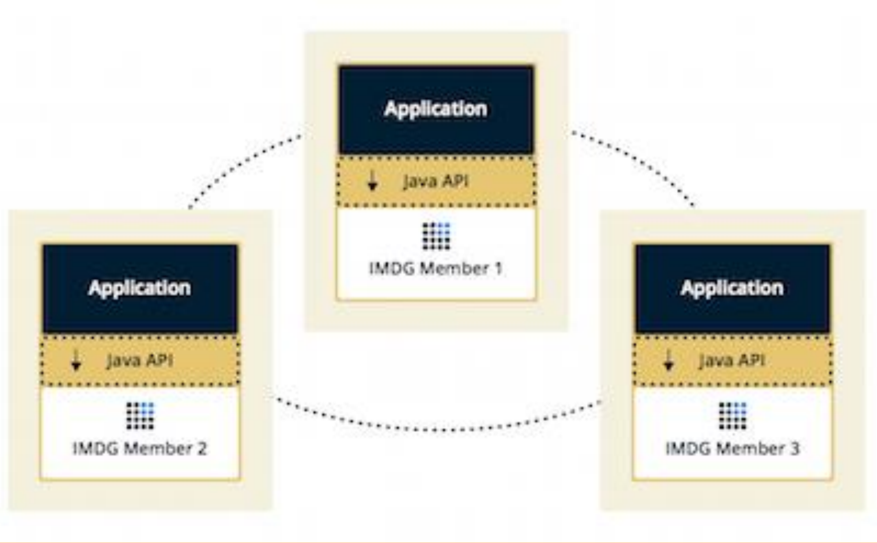
Configuration déclarative : hazelcast.xml



```xml
<hazelcast>
    <group>
        <name>dev</name>
    </group>
    <management-center enabled="false">http://localhost:8080/mancenter</management-center>
    <network>
        <port auto-increment="true" port-count="100">5701</port>
        <join>
            <multicast enabled="true">
                <multicast-group>224.2.2.3</multicast-group>
                <multicast-port>54327</multicast-port>
            </multicast>
            <tcp-ip enabled="false">
                <interface>127.0.0.1</interface>
                <member-list>
                    <member>127.0.0.1</member>
                </member-list>
            </tcp-ip>
        </join>
    </network>
    <map name="default">
        <time-to-live-seconds>0</time-to-live-seconds>
    </map>
</hazelcast>
```

Discovery Mechanisms

- Un cluster Hazelcast est un réseau de membres qui exécutent Hazelcast.
- Un membre peut se joindre automatiquement au cluster pour former un cluster.
- Cette jonction automatique a lieu avec divers mécanismes de découverte que les membres du cluster utilisent pour se retrouver.
  - TCP
  - Multicast (UDP)
  - Cloud Discovery : (AWS, GCP, JClouds, Azure, Zookeeper, PCF, OpenShift, Eureka, Kubernetes, etc.)
- Notez que:
  - Par défaut Hazelcast utilise le mode Multicast (Ce qui est pratique mais déconseillé en production)
  - Dans un cluster, les communications entre ses membres se font toujours via TCP / IP, quel que soit le mécanisme de découverte utilisé.



Data Structures

- Hazelcast has two types of distributed objects in terms of their partitioning strategies:
  1. Data structures where each partition stores a part of the instance, namely partitioned data structures. :
     - **Map**
     - **MultiMap**
     - Cache (Hazelcast JCache implementation)
     - Event Journal
  2. Data structures where a single partition stores the whole instance, namely non-partitioned data structures :
     - **Queue, Set, List, Ringbuffer**
     - Lock, Isemaphore, IAtomicLong, IAtomicReference
     - FlakeIdGenerator, ICountdownLatch, Cardinality Estimator, PN Counter

Data Structures : Exemple d’utilisation de Map et Queue

```
HazelcastInstance hzInstance = Hazelcast.newHazelcastInstance();
```

Distributed Map

```
Map<String, String> capitalcities = hzInstance.getMap( "capitals" );
capitalcities.put( "1", "Rabat" );
capitalcities.put( "2", "Paris" );
```

Distributed Queue

```
BlockingQueue<MyTask> queue = hzInstance.getQueue( "tasks" );
queue.put( new MyTask() );
MyTask task = queue.take();
boolean offered = queue.offer( new MyTask(), 10, TimeUnit.SECONDS );
task = queue.poll( 5, TimeUnit.SECONDS );
if ( task != null ) {
    //process task
}
```

Data Structures : Queue (TaskProducer=>TaskConsumer)

```java
public class ProducerQueueMember {
public static void main( String[] args ) throws Exception {

HazelcastInstance hz = Hazelcast.newHazelcastInstance();

IQueue<Integer> queue = hz.getQueue( "queue" );
        for ( int k = 1; k < 100; k++ ) {
            queue.put( k );
            System.out.println( "Producing: " + k );
            Thread.sleep(1000);
        }
        queue.put( -1 );
        System.out.println( "Producer Finished!" );
    }
}
```

```java
public class ConsumerQueueMember {
    public static void main( String[] args
) throws Exception {
  HazelcastInstance hz =
Hazelcast.newHazelcastInstance();
        IQueue<Integer> queue =
hz.getQueue( "queue" );

while ( true ) {
 int item = queue.take();
System.out.println( "Consumed: " + item );
 if ( item == -1 ) {
    queue.put( -1 );
    break;
   }
   Thread.sleep( 5000 );
        }
System.out.println( "Consumer Finished!" );
    }
}
```

Data Structures : MultiMap

- Hazelcast fournit des implémentation distribuées des structures de données de types :
  - Map
  - Queue
  - **MultiMap**
  - Set
  - List
  - RingBuffer

```
HazelcastInstance hazelcastInstance =
Hazelcast.newHazelcastInstance();
MultiMap<String , String > map =
hazelcastInstance.getMultiMap( "map" );
map.put( "a", "1" );
map.put( "a", "2" );
map.put( "b", "3" );
System.out.println( "PutMember:Done" );


for (String key: map.keySet()){
    Collection<String> values = map.get(key);
    System.out.printf("%s -> %s\n", key, values);
}
```

```
b → [3]
a → [2, 1]
```

Data Structures : Set

- Hazelcast Set does not allow duplicate elements.
- Hazelcast Set does not preserve the order of elements.
- Hazelcast Set is a non-partitioned data structure: all the data that belongs to a set lives on one single partition in that member.
- Hazelcast Set cannot be scaled beyond the capacity of a single machine. Since the whole set lives
- on a single partition, storing a large amount of data on a single set may cause memory pressure.
- Therefore, you should use multiple sets to store a large amount of data. This way, all the sets are spread across the cluster, sharing the load.
- A backup of Hazelcast Set is stored on a partition of another member in the cluster so that data is not lost in the event of a primary member failure.
- All items are copied to the local member and iteration occurs locally.

```java
HazelcastInstance hz =
Hazelcast.newHazelcastInstance();

ISet<String> set = hz.getSet("set");

set.add("Tokyo");
set.add("Paris");
set.add("London");
set.add("New York");
System.out.println("Putting finished!");

for (String value:set){
    System.out.println(value);
}
```

Data Structures : List

- Hazelcast List (IList) is similar to Hazelcast Set, but it also allows duplicate elements.
  - Besides allowing duplicate elements, Hazelcast List preserves the order of elements.
  - Hazelcast List is a non-partitioned data structure where values and each backup are represented by their own single partition.
  - Hazelcast List cannot be scaled beyond the capacity of a single machine.
  - All items are copied to local and iteration occurs locally.

```java
HazelcastInstance hz =
Hazelcast.newHazelcastInstance();

IList<String> myList = hz.getList("myList");

myList.add("Tokyo");
myList.add("Paris");
myList.add("London");
myList.add("New York");
System.out.println("Putting finished!");

for (String value:myList){
    System.out.println(value);
}
```

Data Structures : RingBuffer

- Hazelcast Ringbuffer is a replicated but not partitioned data structure that stores its data in a ringlike structure.
- You can think of it as a circular array with a given capacity. Each Ringbuffer has a tail and a head.
- The tail is where the items are added and the head is where the items are overwritten or expired.
- You can reach each element in a Ringbuffer using a sequence ID, which is mapped to the elements between the head and tail (inclusive) of the Ringbuffer.
- Ringbuffer can sometimes be a better alternative than an Hazelcast IQueue.
- Unlike IQueue, Ringbuffer does not remove the items, it only reads items using a certain position.

```
HazelcastInstance hz =
Hazelcast.newHazelcastInstance();

Ringbuffer<String> ringbuffer =
hz.getRingbuffer("rb");

ringbuffer.add("Item 1");
ringbuffer.add("Item 2");

long sequence = ringbuffer.headSequence();
while(true){
    String item = ringbuffer.readOne(sequence);
    sequence++;
    // process item
}
```

Data Structures : Topic

- Hazelcast provides a distribution mechanism for publishing messages that are delivered to multiple subscribers.
- This is also known as a publish/subscribe (pub/sub) messaging model.
- Publishing and subscribing operations are cluster wide.
- When a member subscribes to a topic, it is actually registering for messages published by any member in the cluster, including the new members that joined after you add the listener.

```java
public class TopicPublisher {
    public static void main(String[] args) {
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();
        ITopic<String> topic = hz.getTopic("topic");
        topic.publish("My message : Hi...");
    }
}
```

```java
public class TopicSubscriber {
    public static void main(String[] args) {
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();
        ITopic<String> topic = hz.getTopic("topic");
        topic.addMessageListener(new MessageListener<String>() {
            @Override
            public void onMessage(Message<String> message) {
  System.out.println("Received message
:"+message.getMessageObject());
            }
        });
        System.out.println("Subscribed");
    }
}
```

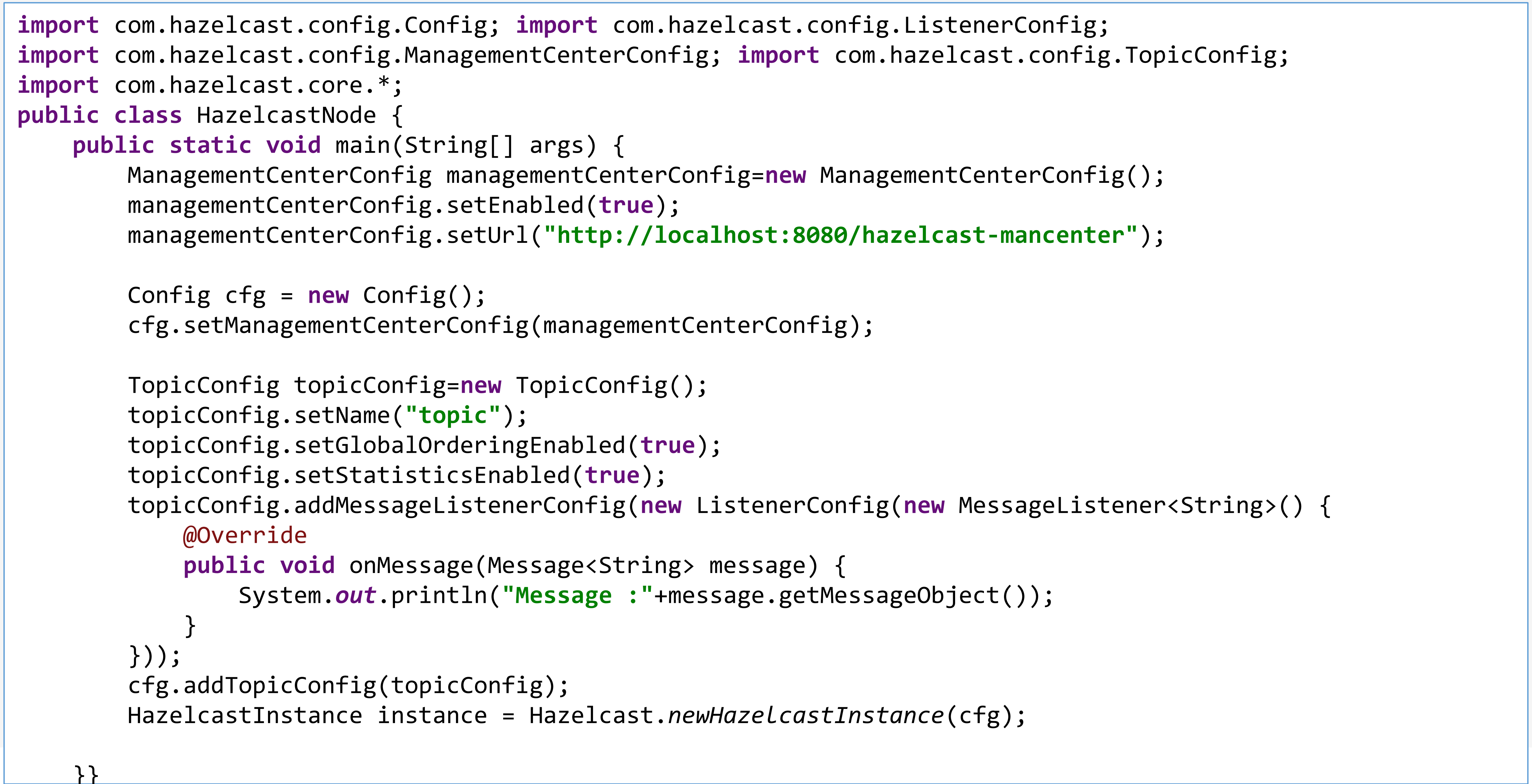
Data Structures : Topic

- Hazelcast provides a distribution mechanism for publishing messages that are delivered to multiple subscribers.
- This is also known as a publish/subscribe (pub/sub) messaging model.
- Publishing and subscribing operations are cluster wide.
- When a member subscribes to a topic, it is actually registering for messages published by any member in the cluster, including the new members that joined after you add the listener.

```
public class TopicPublisher {
    public static void main(String[] args) {
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();
        ITopic<String> topic = hz.getTopic("topic");
        topic.publish("My message : Hi...");
    }
}
```

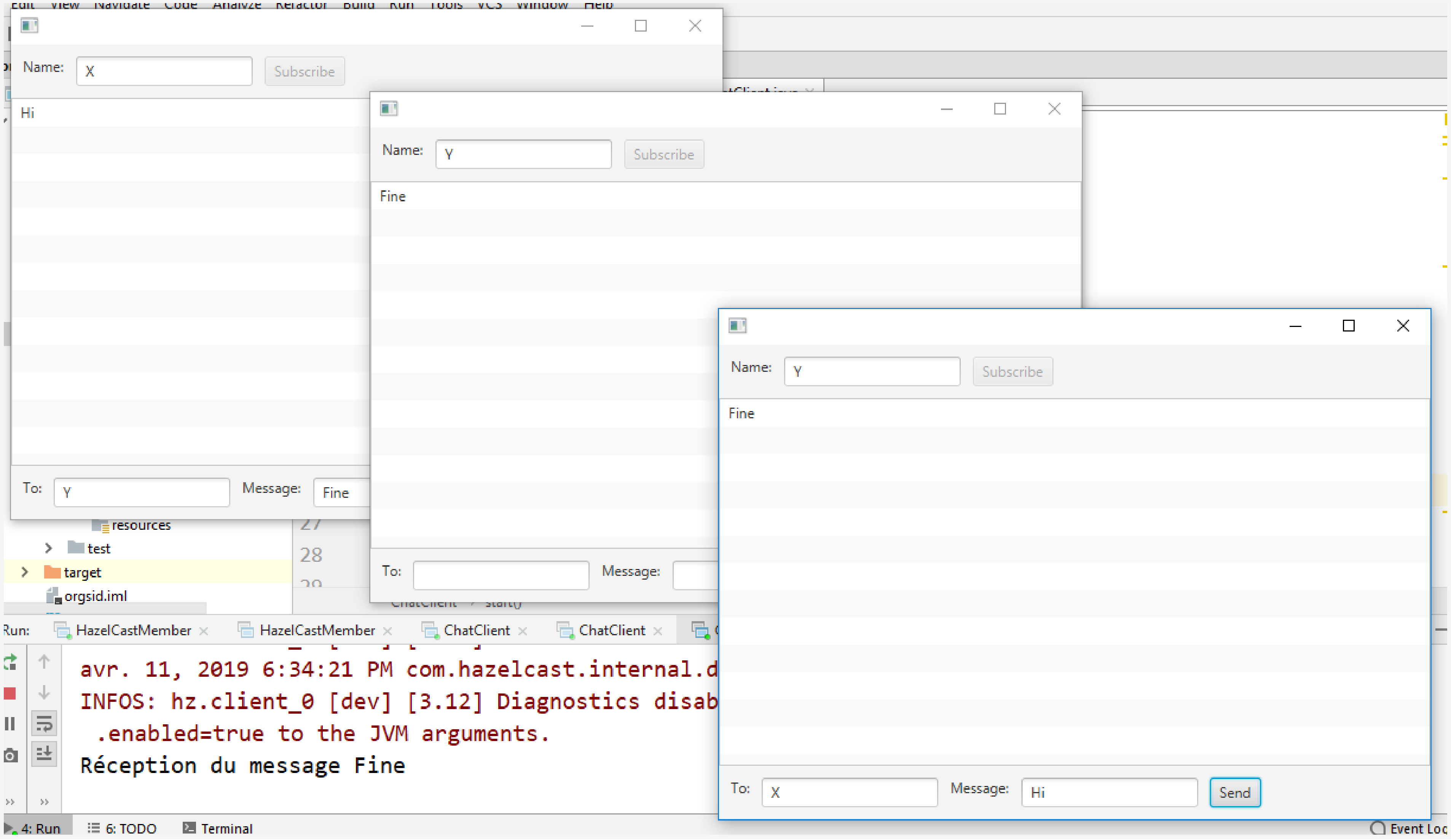
```
public class TopicSubscriber {
    public static void main(String[] args) {
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();
        ITopic<String> topic = hz.getTopic("topic");
        topic.addMessageListener(new MessageListener<String>() {
            @Override
            public void onMessage(Message<String> message) {
  System.out.println("Received message
:"+message.getMessageObject());
            }
        });
        System.out.println("Subscribed");
    }
}
```

Data Structures : Configuring a Topic



```
import com.hazelcast.config.Config; import com.hazelcast.config.ListenerConfig;
import com.hazelcast.config.ManagementCenterConfig; import com.hazelcast.config.TopicConfig;
import com.hazelcast.core.*;
public class HazelcastNode {
    public static void main(String[] args) {
        ManagementCenterConfig managementCenterConfig=new ManagementCenterConfig();
        managementCenterConfig.setEnabled(true);
        managementCenterConfig.setUrl("http://localhost:8080/hazelcast-mancenter");

        Config cfg = new Config();
        cfg.setManagementCenterConfig(managementCenterConfig);

        TopicConfig topicConfig=new TopicConfig();
        topicConfig.setName("topic");
        topicConfig.setGlobalOrderingEnabled(true);
        topicConfig.setStatisticsEnabled(true);
        topicConfig.addMessageListenerConfig(new ListenerConfig(new MessageListener<String>() {
            @Override
            public void onMessage(Message<String> message) {
                System.out.println("Message :"+message.getMessageObject());
            }
        }));
        cfg.addTopicConfig(topicConfig);
        HazelcastInstance instance = Hazelcast.newHazelcastInstance(cfg);

    }}
```

Data Structures : Client Chat avec Hazelcast Topic



Name: X

Hi

To: Y Message: Fine

Name: Y

Fine

To: Message:

Name: Y

Fine

To: X Message: Hi

```
avr. 11, 2019 6:34:21 PM com.hazelcast.internal.d
INFOS: hz.client_0 [dev] [3.12] Diagnostics disab
 .enabled=true to the JVM arguments.
Réception du message Fine
```

Data Structures : Client Chat avec Hazelcast Topic

```java
import com.hazelcast.client.HazelcastClient;import com.hazelcast.client.impl.clientside.HazelcastClientInstanceImpl;
import com.hazelcast.core.Hazelcast; import com.hazelcast.core.HazelcastInstance;
import com.hazelcast.core.ITopic; import javafx.application.Application; import javafx.collections.FXCollections;
import javafx.collections.ObservableArray; import javafx.collections.ObservableList;import javafx.geometry.Insets;
import javafx.scene.Scene;import javafx.scene.control.*; import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox; import javafx.stage.Stage; import java.util.ArrayList; import java.util.List;
public class ChatClient extends Application {
    private HazelcastInstance hazelcastInstance;
    @Override
    public void start(Stage primaryStage) throws Exception {
        hazelcastInstance= HazelcastClient.newHazelcastClient();
        BorderPane borderPane=new BorderPane();
        Label labelName=new Label("Name:");
        TextField textFieldName=new TextField();
        Button buttonSubscribe=new Button("Subscribe");
        Label labelTo=new Label("To:");
        TextField textFieldTo=new TextField();
        Button buttonSend=new Button("Send");
        Label labelMessage=new Label("Message:");
        TextField textFieldMessage=new TextField();
        List<String> messages=new ArrayList<>();
        ObservableList<String> observableList= FXCollections.observableList(messages);
        ListView<String> listView=new ListView<>(observableList);
```

Data Structures : Client Chat avec Hazelcast Topic

```
HBox hBoxTop=new HBox(10); hBoxTop.setPadding(new Insets(10));
        hBoxTop.getChildren().addAll(labelName,textFieldName,buttonSubscribe);
        HBox hBoxBottom=new HBox(10);hBoxBottom.setPadding(new Insets(10));
        hBoxBottom.getChildren().addAll(labelTo,textFieldTo,labelMessage,textFieldMessage,buttonSend);
        borderPane.setTop(hBoxTop);
        borderPane.setBottom(hBoxBottom);
        borderPane.setCenter(listView);
        Scene scene=new Scene(borderPane,600,400);
        primaryStage.setScene(scene);
        primaryStage.show();

        buttonSubscribe.setOnAction(evt->{
            ITopic<String> topic=hazelcastInstance.getTopic(textFieldName.getText());
            topic.addMessageListener(message->{
                System.out.println("Réception du message "+message.getMessageObject());
                observableList.add(message.getMessageObject());
            });
            buttonSubscribe.setDisable(true);
        });

        buttonSend.setOnAction(evt->{
            ITopic topicTo=hazelcastInstance.getTopic(textFieldTo.getText());
            topicTo.publish(textFieldMessage.getText());
        });
    }
}
```

Loading and Storing Persistent Data

- Hazelcast allows you to load and store the distributed Structures entries from/to a persistent data store such as a relational database.
- To do this, for example in Map Structure, you can use Hazelcast’s MapStore and MapLoader interfaces.
- When you provide a MapLoader implementation and request an entry (IMap.get()) that does not exist in memory, MapLoader's load method loads that entry from the data store.
- This loaded entry is placed into the map and will stay there until it is removed or evicted.
- When a MapStore implementation is provided, an entry is also put into a user defined data store.
- Data store needs to be a centralized system that is accessible from all Hazelcast members. Persistence to a local file system is not supported.

Loading and Storing Persistent Data

```java
package a;
import com.hazelcast.core.MapStore; import java.util.Collection;
import java.util.Map;
public class PersonMapStore implements MapStore<String,Person> {
    @Override
    public void store(String s, Person person) {
        System.out.println("***************");
        System.out.println("Storing Persone "+person.getName());
    }
    @Override
    public void storeAll(Map<String, Person> map) {    }
    @Override
    public void delete(String s) {    }
    @Override
    public void deleteAll(Collection<String> collection) {    }
    @Override
    public Person load(String s) {  return null;  }
    @Override
    public Map<String, Person> loadAll(Collection<String> collection) {
        return null;
    }
    @Override
    public Iterable<String> loadAllKeys() {
        return null;
    }
}
```

```java
package a;

import java.io.Serializable;
public class Person implements Serializable {
    private Long id;
    private String name;

    public Person(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Loading and Storing Persistent Data

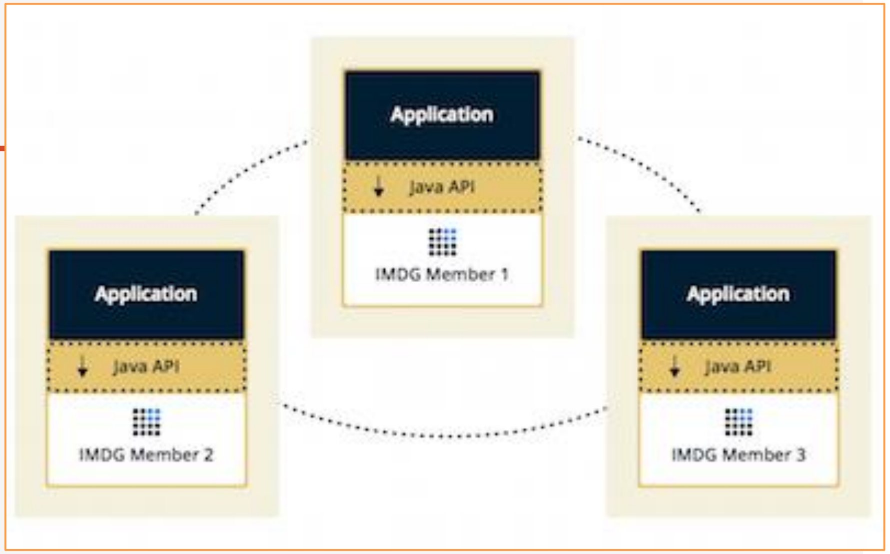
```
Config config = new Config();
MapConfig mapConfig = config.getMapConfig("default");
mapConfig.setName("MyQueue")
        .setBackupCount(1)
        .setStatisticsEnabled(true);

mapConfig.getMapStoreConfig()
        .setEnabled(true)
        .setClassName("a.PersonMapStore")
        .setProperty("binary", "false");
config.addMapConfig(mapConfig);
HazelcastInstance hz=Hazelcast.newHazelcastInstance(config);

Map<String, Person> capitalcities = hz.getMap( "MyQueue" );
capitalcities.put( "1", new Person(1L,"Mohamed") );
capitalcities.put( "1", new Person(2L,"Imane") );
```

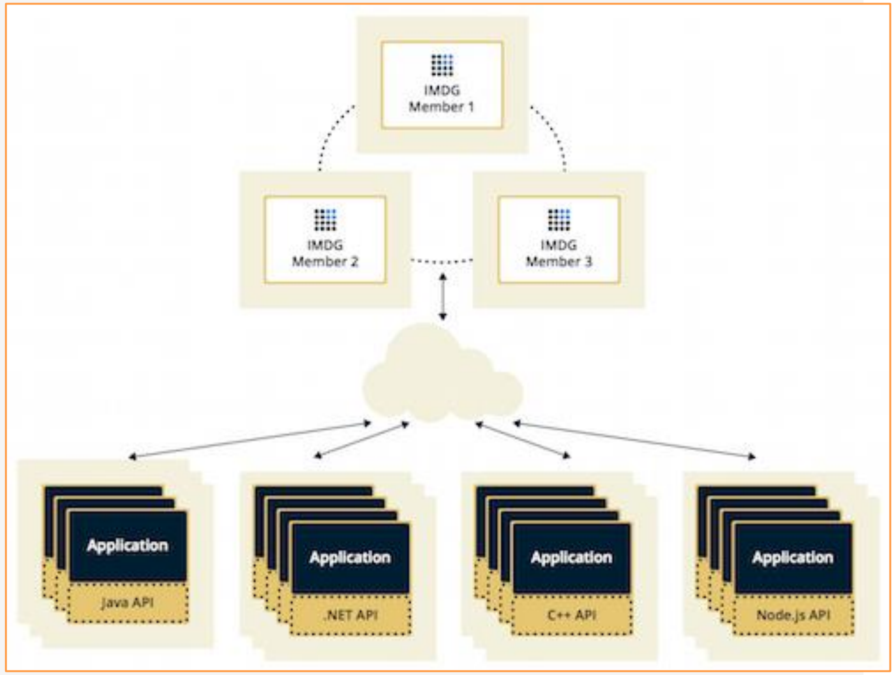
Data Structures

- Hazelcast fournit des implémentation distribuées des structures de données de types :
    - Map
    - Queue
    - MultiMap
    - Set
    - List
    - RingBuffer

Distributed Computing with Hazelcast

- Hazelcast offers IExecutorService for you to use in distributed environments.
- It implements java.util.concurrent.ExecutorService to serve the applications requiring computational and data processing power.
- With IExecutorService, you can execute tasks asynchronously and perform other useful tasks.
- If your task execution takes longer than expected, you can cancel the task execution.
- Tasks should be Serializable since they are distributed.
- In the Java Executor framework, you implement tasks two ways: Callable or Runnable.
  - Callable: If you need to return a value and submit it to Executor, implement the task as **java.util.concurrent.Callable**.
  - Runnable: If you do not need to return a value, implement the task as **java.util.concurrent.Runnable**.



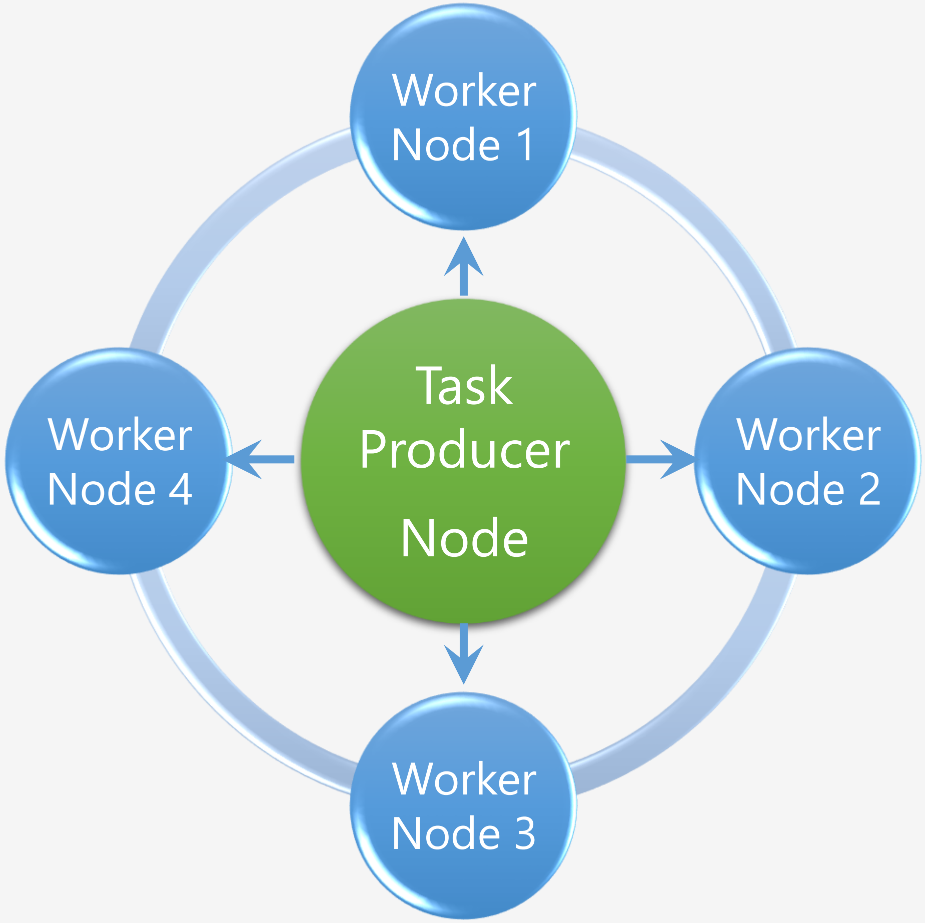


Distributed Computing with Hazelcast : Callable Task

```java
import com.hazelcast.core.HazelcastInstance;
import com.hazelcast.core.HazelcastInstanceAware;
import com.hazelcast.core.IMap;

import java.io.Serializable;
import java.util.concurrent.Callable;

public class SumTask implements Callable<Integer>, Serializable, HazelcastInstanceAware {
    private transient HazelcastInstance hazelcastInstance;
    @Override
    public void setHazelcastInstance(HazelcastInstance hazelcastInstance) {
        this.hazelcastInstance=hazelcastInstance;
    }
    @Override
    public Integer call() throws Exception {
        IMap<Integer, Integer> map = hazelcastInstance.getMap( "inputData" );
        int result = 0;
        for ( Integer key : map.localKeySet() ) {
            System.out.println( "Calculating for key: " + key );
            result += map.get( key );
        }
        System.out.println( "Local Result: " + result );
        return result;
    }
}
```

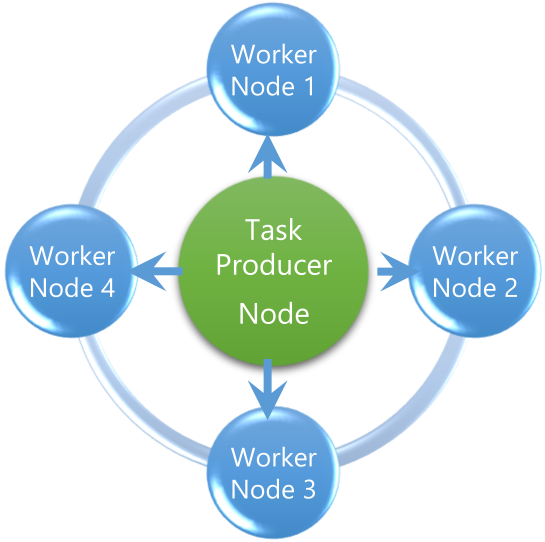
Distributed Computing with Hazelcast : Callable Task

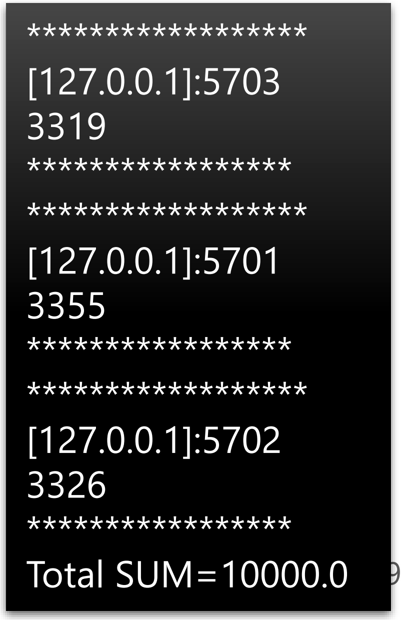
```
import com.hazelcast.config.Config;
import com.hazelcast.config.ManagementCenterConfig;
import com.hazelcast.core.Hazelcast;
import com.hazelcast.core.HazelcastInstance;

public class HazelCastWorker {
    public static void main(String[] args) {
        Config config=new Config();
        ManagementCenterConfig centerConfig=new
ManagementCenterConfig();
        centerConfig.setEnabled(true);
        centerConfig.setUrl("http://localhost:8080/hazelcast-
mancenter");
        config.setManagementCenterConfig(centerConfig);
        HazelcastInstance hazelcastInstance=
Hazelcast.newHazelcastInstance();
    }
}
```



Distributed Computing with Hazelcast : Callable Task

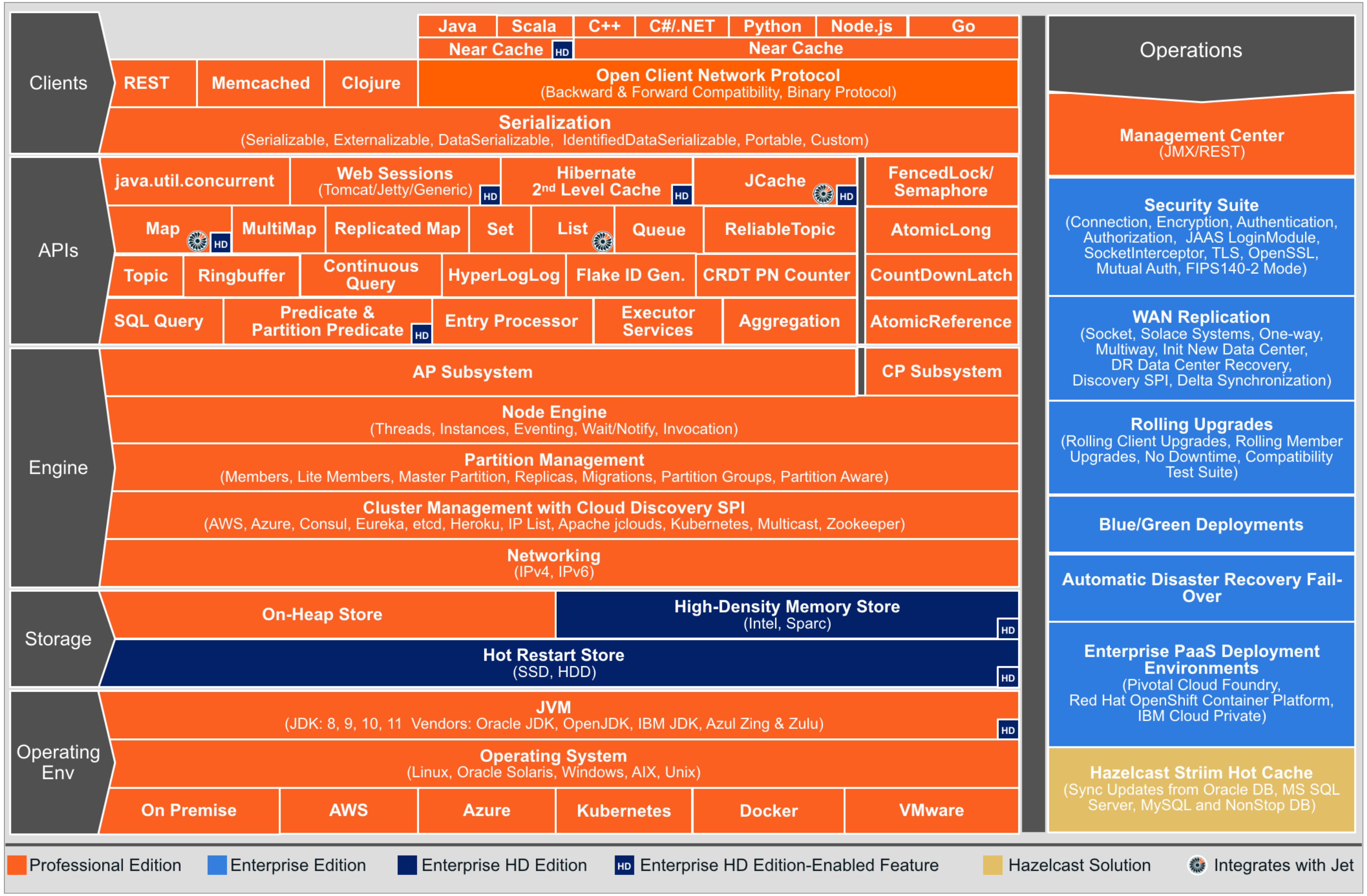
```
import com.hazelcast.client.HazelcastClient; import com.hazelcast.core.HazelcastInstance;
import com.hazelcast.core.IExecutorService;import com.hazelcast.core.IMap; import com.hazelcast.core.Member;
import java.util.Map; import java.util.concurrent.ExecutionException; import java.util.concurrent.Future;
public class TeskProducer {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        HazelcastInstance hazelcastInstance= HazelcastClient.newHazelcastClient();
        IMap<Integer,Integer> data=hazelcastInstance.getMap("inputData");
        for (int i = 0; i <10000 ; i++) {  data.put(i,1);  }
        IExecutorService executorService=hazelcastInstance.getExecutorService("default");
        /*
        Future<Integer> response=executorService.submit(new SumTask());
        System.out.println("Result="+response.get());
        */
        Map<Member,Future<Integer>> response=
executorService.submitToAllMembers(new SumTask());
        double reduceSum=0;
        for (Member member:response.keySet()){
            System.out.println("******************");
            System.out.println(member.getAddress());
            System.out.println(response.get(member).get());
            reduceSum+=response.get(member).get();
            System.out.println("*****************");
        }
        System.out.println("Total SUM="+reduceSum);
    }
}
```





```
******************
[127.0.0.1]:5703
3319
*****************
******************
[127.0.0.1]:5701
3355
*****************
******************
[127.0.0.1]:5702
3326
*****************
Total SUM=10000.0
```

Architecture de Hazecast



Clients

Java | Scala | C++ | C#/.NET | Python | Node.js | Go

Near Cache HD | Near Cache

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) HD | Hibernate 2nd Level Cache HD | JCache HD | FencedLock/ Semaphore

Map HD | 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 HD | Entry Processor | Executor Services | Aggregation | AtomicReference

Engine

AP Subsystem | CP Subsystem

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) HD

Hot Restart Store
(SSD, HDD) HD

Operating Env

JVM
(JDK: 8, 9, 10, 11 Vendors: Oracle JDK, OpenJDK, IBM JDK, Azul Zing & Zulu) HD

Operating System
(Linux, Oracle Solaris, Windows, AIX, Unix)

On Premise | AWS | Azure | Kubernetes | Docker | VMware

Operations

Management Center
(JMX/REST)

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

Enterprise PaaS Deployment Environments
(Pivotal Cloud Foundry, Red Hat OpenShift Container Platform, IBM Cloud Private)

Hazelcast Striim Hot Cache
(Sync Updates from Oracle DB, MS SQL Server, MySQL and NonStop DB)

🟧 Professional Edition 🟦 Enterprise Edition ■ Enterprise HD Edition HD Enterprise HD Edition-Enabled Feature 🟨 Hazelcast Solution ⚙ Integrates with Jet