




**Store Distribuido**



La idea al crear hadoop era poder crear indices invertidos (para google por ejemplo).  
Entonces uno busca algo y te devuelve una lista de paginas en la que esta.

# HDFS

Hadoop Distributed File System



Hadoop salio de un paper.

La idea de esta herramienta es que sea transparente para el usuario que el procesamiento se esta haciendo de manera distribuida.

Entonces lo que hicieron fue separarlo en partes. Una parte maneja recursos y tareas (genericas) distribuidas, y map-reduce hace uso de YARN.

## Hadoop: componentes

**Hadoop Distributed  
File System (HDFS™):**  
File System distribuido



**Hadoop MapReduce**  
Framework de  
procesamiento  
distribuido



**Hadoop YARN**

Framework de manejo de  
recursos y tareas  
distribuidas






# HDFS: Características.

- **File system distribuido** modelado a partir del paper [Google File System](#).
- Optimizado para **high throughput** y trabaja mejor **leyendo y escribiendo archivos muy grandes** (órdenes de Gigas o más)
- Utiliza **tamaños de bloque muy grande**.
- Maneja **replicación para lograr fault tolerance**.
- Diseñado para correr en **commodity hardware** (no RAID, NFS, etc.)

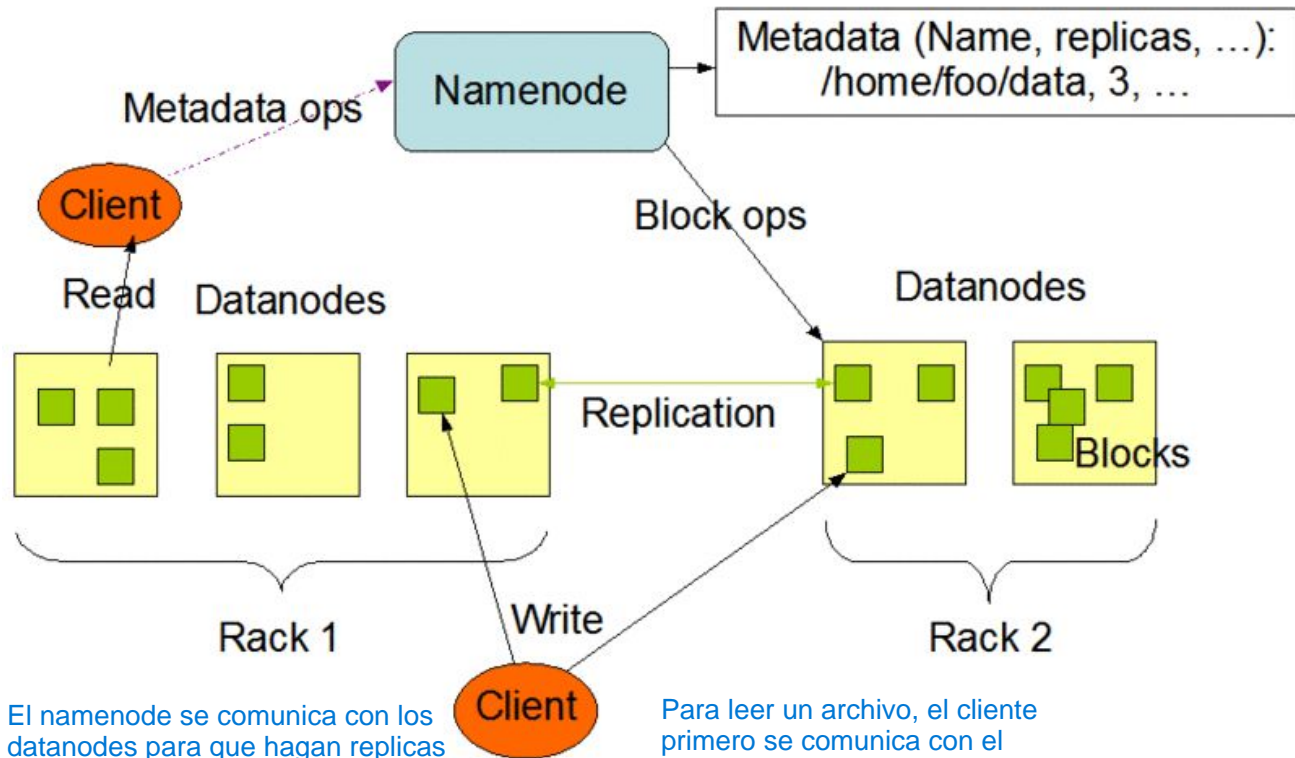
La idea de commodity hardware es no tener que hacer escalamiento vertical (con computadoras super potentes). Cualquier computadora gama media funciona bien.



# Hadoop: arquitectura

Esta es master-slave. Cuando uno baja hadoop, tiene que correr dos aplicaciones diferentes. Una de master y una de slave.

## HDFS Architecture



El namenode se comunica con los datanodes para que hagan replicas

Para leer un archivo, el cliente primero se comunica con el namenode. El namenode le contesta donde esta cada parte de ese archivo.  
En hadoop me permite leer de replicas (pero no me entero)


(namenode = master)  
Recibe todos los pedidos de los clientes (como crear X carpeta o guardar Y archivo en Z carpeta.  
Guarda la informacion en el filesystem.  
El namenode le manda estos pedidos a los datanodes.  
El cliente simplemente hace put al namenode, sin enterarse de todo este procesamiento paralelo.  
Lo unico que ve el cliente desde afuera es un filesystem.  
Todo el resto de los componentes los ve el namenode.  
El namenode no recibe el archivo en ningun momento, sino que le dice al cliente a quien mandarle el pedido.

Cada datanode no sabe que partes de que archivos guarda, a el le dicen "guarda esto" y el solo lo guarda.  
El namenode es el que sabe donde esta cada cosa.



# HDFS: Bloque

El bloque es la unidad de información que se traslada

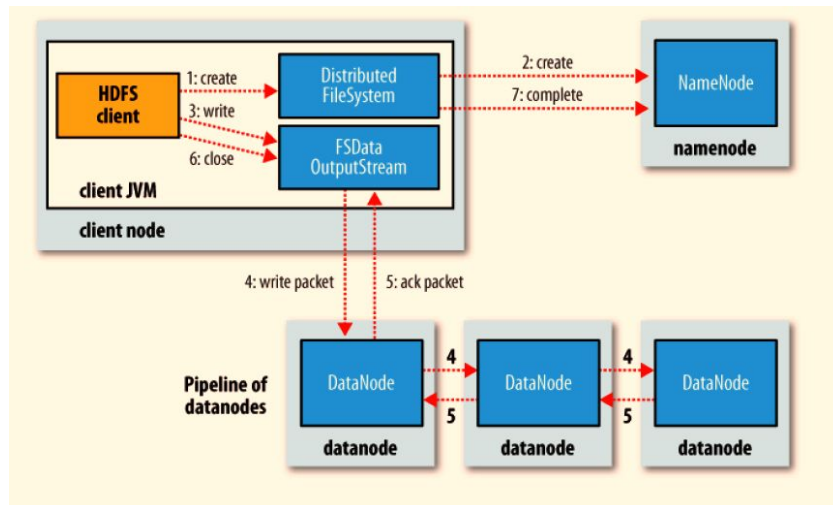
- Los archivos en HDFS se particiones **en bloques de tamaño definido en configuración** (por defecto de **128MB**)
  - Cada bloque es **replicado** en diferentes nodos para redundancia. La cantidad de réplicas es configurable
  - Si el **archivo es menor que un bloque, solo ocupa su tamaño real.** (porque en muchos filesystems si guardo algo menor que el tam de bloque pierdo todo ese espacio)
  - Al particionar en bloques **se simplifica el manejo de archivos** porque al tener tamaño máximo un nodo puede manejar un bloque.
- 

# HDFS: Name Node

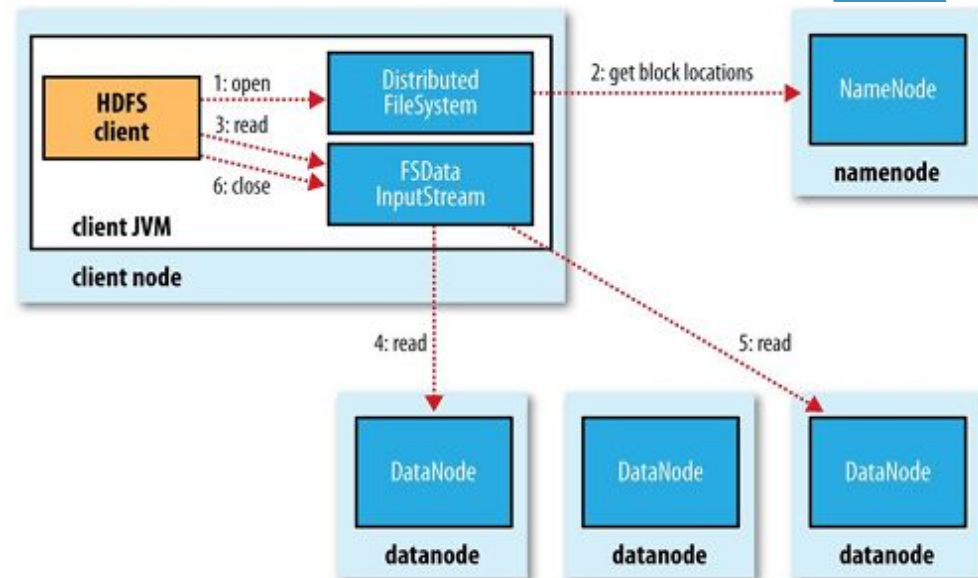
Originalmente, el namenode era SPOF. La primera solución era tener un disco pasivo que estaba conectado por red.

- Administra el **namespace del file system** y regula el acceso de los clientes.
- Administra y mantiene la estructura lógica y metadata de carpetas y archivos (permisos, tamaños y ubicaciones). Usa la estructura Linux.
- Relaciona archivos con sus bloques
- **Conoce el estado de los Data Nodes**
- Responde los pedidos de búsqueda y escrituras de bloques.

El cliente se comunica con el namenode para decirle que guarde algo. El namenode le dice que lo escriba a un determinado datanode. El cliente lo guarda, y le avisa al namenode que lo guardó.



Aca el cliente quiere leer algo. El namenode le contesta en que namenodes esta cada parte de cada archivo. El cliente le hace esos pedidos a los datanodes.



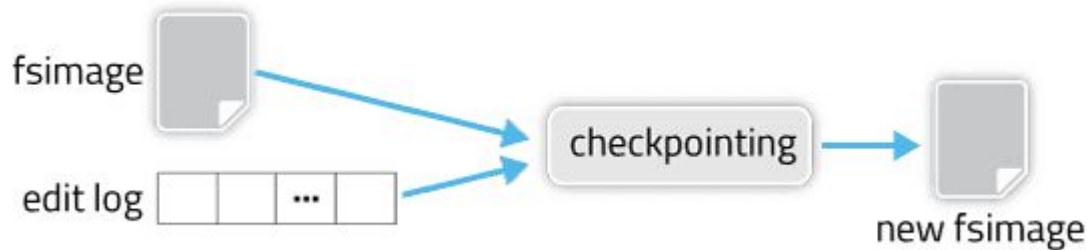
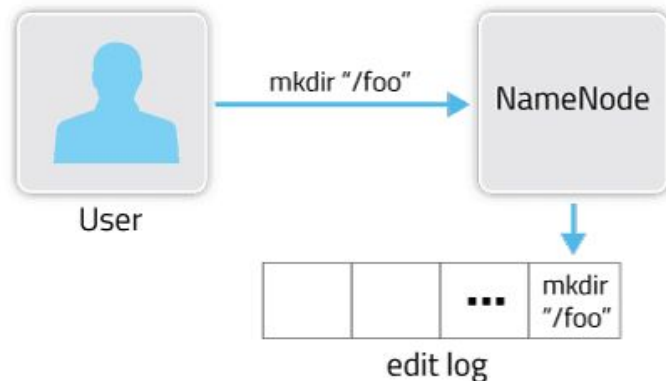
# HDFS: Name Node

➤ **Persiste la información en el disco.** A partir de dos componentes

- FsImage: Snapshot que se toma cada tanto del estado del file system
- Transaction log: estructura que guarda las escrituras hasta que la FsImage se actualice.

Para poder trabajar lo menos posible, la idea es usar el disco lo menos posible. Mientras mas cosas tenga en memoria mejor. La idea es que yo voy a tener en memoria un log de las operaciones.

Cada tanto esta estructura baja al disco (checkpointing), y vacio lo que tengo en memoria. Entonces si ya encuentro la respuesta a una lectura antes de ir a disco, la devuelvo mas rapido. NO ES un cache de respuestas. Es un log de las operaciones.









## HDFS: **Data Nodes.**

Los datanodes tienen 3 responsabilidades

- **Guarda los bloques de información** que son parte de los archivos guardados en el sistema.
  - **Se reporta al Name Node** periódicamente (default 3 segundos) para informar que sigue vivo y cuales son los bloques que tiene.
  - **Se comunica con los otros Data nodes** para transmitir la información que se está replicando un nodo.
- 



# HDFS: Problemas

- **El Name Node es un SPOF** (single point of failure)
    - Primera aproximación de solución en HDFS 2.X con “federación”
    - Solución definitiva cuando se agregó HA en 3.X.
  - **Problemas con el acceso random a la información de los archivos**
    - Cómo está optimizado para leer por bloque, acceder a un byte en el medio no es la mejor manera de usarlo.
    - Se crearon bases de datos como HBASE, para agregarle una capa de acceso al HDFS.
  - **Muchos archivos pequeños degradan la performance.**
    - Se deben intentar evitar, si es posible mergear o particionar de manera que no afecte a la performance.
- 

Hadoop recibe un parametro, que es a quien quiero pegarle. En este caso es el filesystem, pero podría ser otra cosa (como map-reduce)

## ¿Cómo se usa?

```
$> hadoop fs -command <args>
```

```
$> hadoop fs -get hdfs://nn.example.com/user/hadoop/file localfile #Copy files to the local system.
$> hadoop fs -put -f localfile1 localfile2 /user/hadoop/hadoopdir #Copy from local to destination system
$> hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 #copy from source to destination.
$> hadoop fs -mv hdfs://nn.example.com/file1 hdfs://nn.example.com/file2 #Moves from source to destination.
$> hadoop fs -mkdir /user/hadoop/dir1 #creates directories.
$> hadoop fs -ls /user/hadoop/file1 #returns list of its direct children
$> hadoop fs -rm hdfs://nn.example.com/file /user/hadoop/emptydir #Delete files specified.
$> hadoop fs -rmdir /user/hadoop/emptydir #Delete a directory.
$> hadoop fs -tail pathname #Displays last kilobyte of the file to stdout.
$> hadoop fs -getmerge -nl /src /opt/output.txt #Takes a source directory and a destination file as input and concatenates files in src into the destination local file.
```

[Más comandos](#)

Getmerge agarra todos los archivos de un directorio y los mergea (concatena) en uno solo




# Hazelcast



# Hazelcast

Basicamente es una base de datos de objetos. Distribuido porque puedo tener varios nodos funcionando como uno solo. Su principal competencia es Redis

**Es un grid de datos en memoria, distribuido y open source.**  
**Realizado en Java, con mínimas dependencias y la posibilidad de correr embebido o mediante una arquitectura client-server**





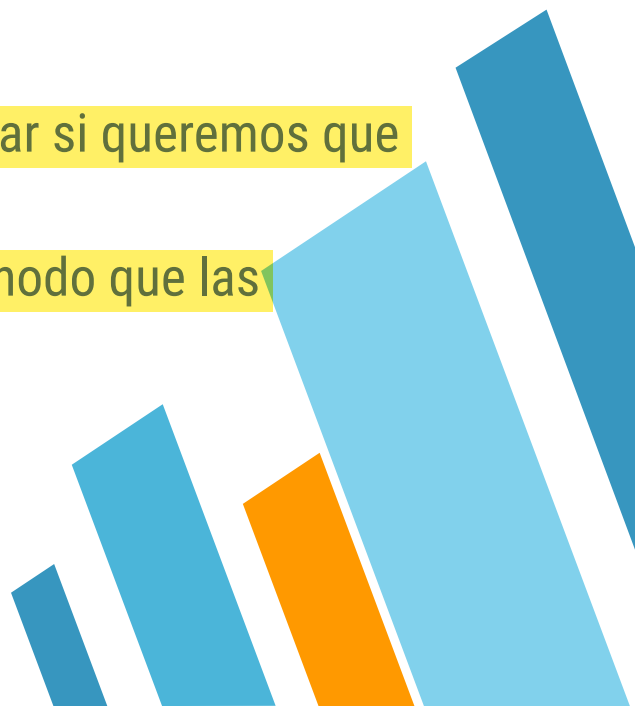
# Hazelcast- Modelo de datos

Hazelcast **redefine varias estructura de datos**, respetando la interfaz conocida de Java para distribuir colecciones:

- IMap
  - IList
  - IQueue
  - ITopic
- Dentro de Hazelcast, voy a tener colecciones que mapean a las de java.

Estas estructuras de datos son las que tenemos que usar si queremos que la data sea distribuida.

Si usamos las estructuras tradicionales sólo las usa el nodo que las declara.





# Hazelcast - Casos de Uso

Principales casos de uso en los cuales Hazelcast puede ser útil:

- In-Memory Data Grid
  - Caching
  - Microservices (tambien hay locks distribuidos)
  - Web Session Clustering
  - Messaging (es lo de los topicos, que son parecidas a las queues)
  - In-Memory NoSQL
  - Application Scaling
  - Procesamiento de operaciones distribuidos
- 

# Ejercicio 1 - Levantar un cluster

Hazelcast trabaja con un jar simple sin otras dependencias para levantar un nodo.

1. Bajar de campus el archivo **hazelcast-all-3.8.6.jar** en algún directorio creado para utilizar con hazelcast.
2. Setear el classpath apuntando al jar.
3. Correr la aplicación en 2 consolas.

```
$> export CLASSPATH=$HOME/hazel/hazelcast-all-3.8.6.jar
```

```
$> java com.hazelcast.console.ConsoleApp
```

0

```
$> java -jar hazelcast-all-3.8.6.jar
```



# Hazelcast - Uso

El nodo se inicializa y a medida que se van incorporando los miembros aparecen en el listado y se puede utilizar la consola para realizar operaciones.

```
INFO: [192.168.1.103]:5701 [dev] [3.5.2]
```

```
Members [2] {  
    Member [192.168.1.103]:5702  
    Member [192.168.1.103]:5701 this  
}
```

```
...
```

```
hazelcast[default] >
```

## Ejercicio 2 - Uso

1. Desde la consola el comando **help** permite ver cuáles son los comandos a operar.
2. El **[default]** indica el *namespace* (nombre de la colección que estamos usando).
3. Insertar en el mapa para la clave *alumno* su legajo y luego obtener el valor de dicha clave.

```
hazelcast[default] > m.put  alumno <legajo>  
hazelcast[default] > m.get  alumno
```

4. Cambiar el *namespace* al legajo para no pisarse y repetir

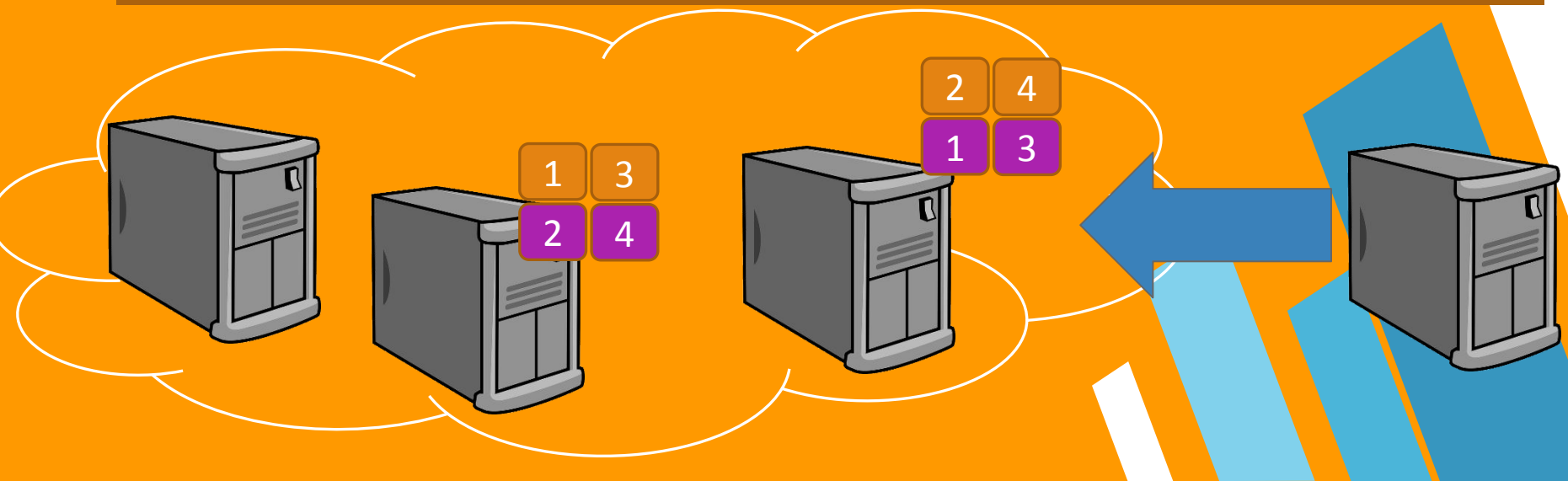
```
hazelcast[default] > ns <legajo>  
hazelcast[<legajo>] > ...
```

# Hazelcast - Tipos de nodo

**Node Member:** Es el tipo de nodo por *default*. Estos tienen datos distribuidos y además realizan tareas distribuidas.

**Lite Member:** Son nodos que no guardan información, sirven para lanzar tareas y otro tipo de acciones (como agregar *listeners*).

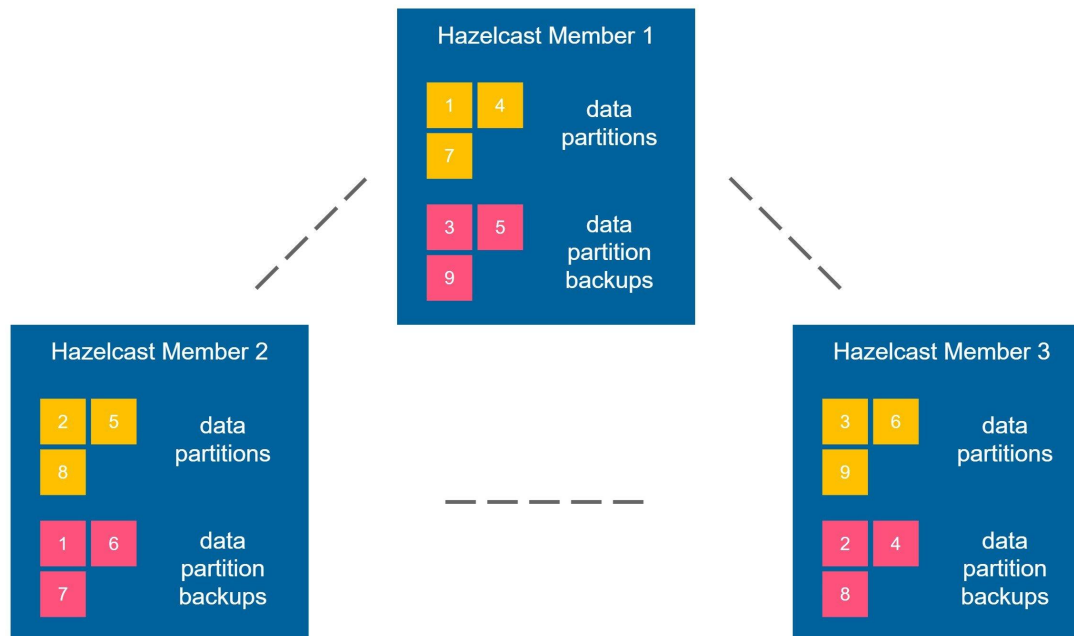
**Native Client:** No es un nodo del cluster, sino una manera en la que me puedo conectar a un cluster de manera Cliente/Servidor. Este cliente se utiliza para poner datos distribuidos dentro del cluster, pedir ejecutar cosas en el cluster.



# Hazelcast - Coordinación

El cluster de Hazelcast tiene una arquitectura **peer-to-peer**, no hay un *master*, sino que cualquier nodo puede asumir el control. Entre otros beneficios elimina la posibilidad de un Single Point Of Failure.

Para la coordinación y asignación de trabajo y particiones se **selecciona al nodo de mayor antigüedad del cluster como Coordinador**.

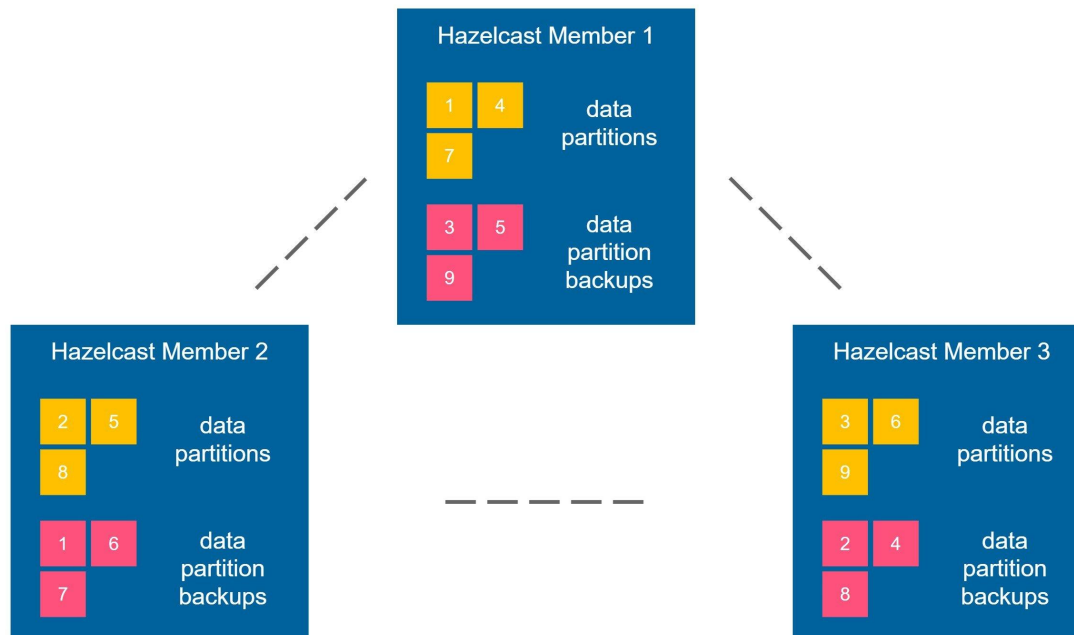


# Hazelcast - Coordinador

Esta info igual se va repartiendo entre todos los nodos por si se cae el master, pero no implica que todos los nodos tengan toda la info en todo momento.

El coordinador **posee una tabla de particiones donde dice qué nodo es owner/backup** de qué particiones, pero la distribuye entre todos. O sea, al llegar/desaparecer un nodo la actualiza y la vuelve a distribuir.

En caso de que el coordinador se caiga se selecciona al siguiente nodo más viejo como nuevo coordinador y se recalcula la tabla de particiones.

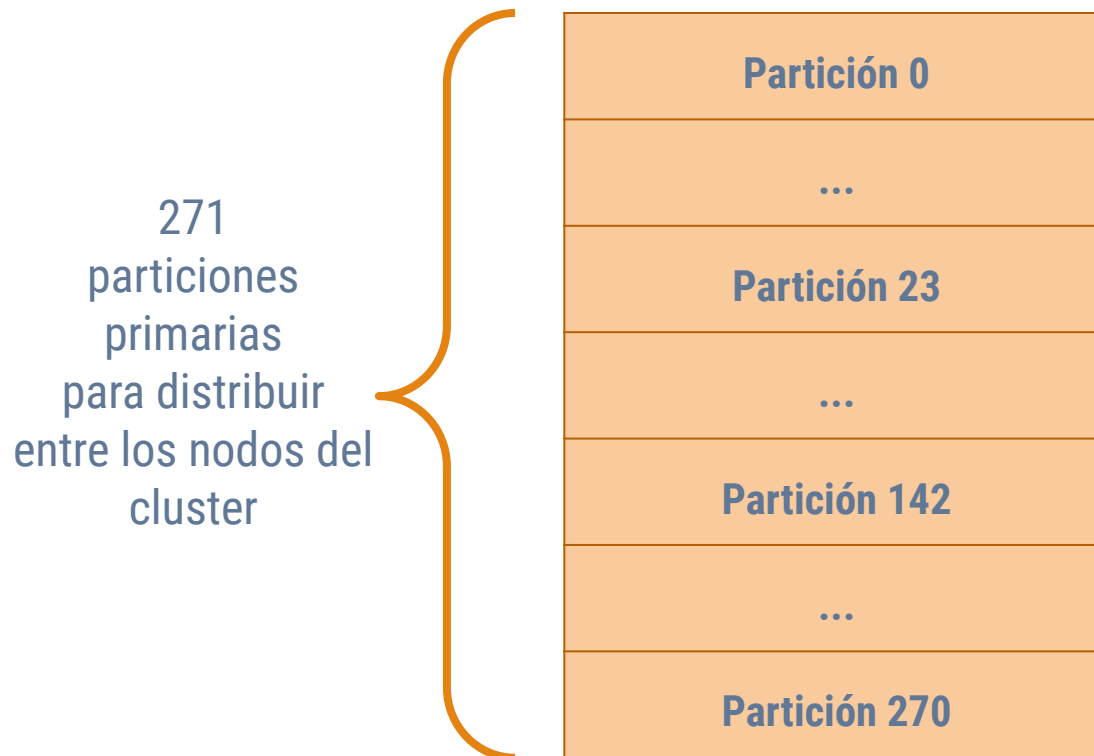


# Hazelcast - Particionado

Cada particion puede tener n copias

La estructura global se divide **en 271 particiones** entre los nodos del cluster, con **posibilidad de redundancia (réplicas o backups) entre nodos**.

Aunque los datos están distribuidos entre nodos, cualquier nodo accede a la totalidad de los datos (si no los tiene él igual puede solicitar a otros nodos).





# Hazelcast - Asignación a Particiones

Para decidir qué partición es owner de qué objeto, usa el algoritmo “Consistent Hashing”.

$$\text{NroParticion(objeto)} = \text{hash(objeto)} \% 271$$


# Hazelcast - Asignación a Particiones

Partición 0:

...

Partición 23

materia<"72.23", "Compiladores">

...

Partición 142

materia<"72.02", "Embedded Systems">

materia<"72.14", "POD">

...

Partición 270

Suponiendo las siguientes inserciones:

```
Map<String, String> materias =  
hz.getMap("materias");  
  
materias.put("72.02", "Embedded Systems");  
  
materias.put("72.23", "Compiladores");  
  
materias.put("72.14", "POD");
```

$\text{haschode}("72.02") \% 271 = 142$

$\text{haschode}("72.23") \% 271 = 23$

$\text{haschode}("72.14") \% 271 = 142$





# Hazelcast - Réplicas

**¿Si hay un solo nodo, hay backup?**

Rta: No

**¿Si hubiera 2 nodos y backup, cuántas particiones les toca a cada uno de esos 2 nodos?**

Rta: aprox.  $271/2$  primarias y  $271/2$  backup (del otro nodo)


**¿Si hubieran 3 nodos, cuántas particiones les tocaría a cada uno de esos 3 nodos?**

Rta: aprox.  $271/3$  primarias y  $271/3$  backup (de los otros nodos)






# Hazelcast - Réplicas

- Las **réplicas se actualizan sincrónicamente** con la partición principal.
  - Por default, **hay una única réplica** porque ese sincronismo provoca una caída en la performance pero **la cantidad de réplicas puede aumentarse o reducirse**.
- 



# Hazelcast - Réplicas Recuperero

- Ante la **caída del nodo que es owner** de cierta partición (tiene la copia principal), **el nodo que contiene su réplica pasa a ser el nuevo owner y su réplica pasa a ser la principal**. En dicho caso otro nodo será asignado para mantener una réplica de la partición.
  - Al agregarse o retirarse nodos se inicia la migración de las particiones, de manera que nunca principal y réplica queden en el mismo nodo.
- 

# Hazelcast - Migración de Particiones

Dado el caso anterior, si agregamos un nodo nuevo:

Partición 0:

...

Partición 23

```
materia<"72.02", "Embedded Systems">
```

...

Partición 142

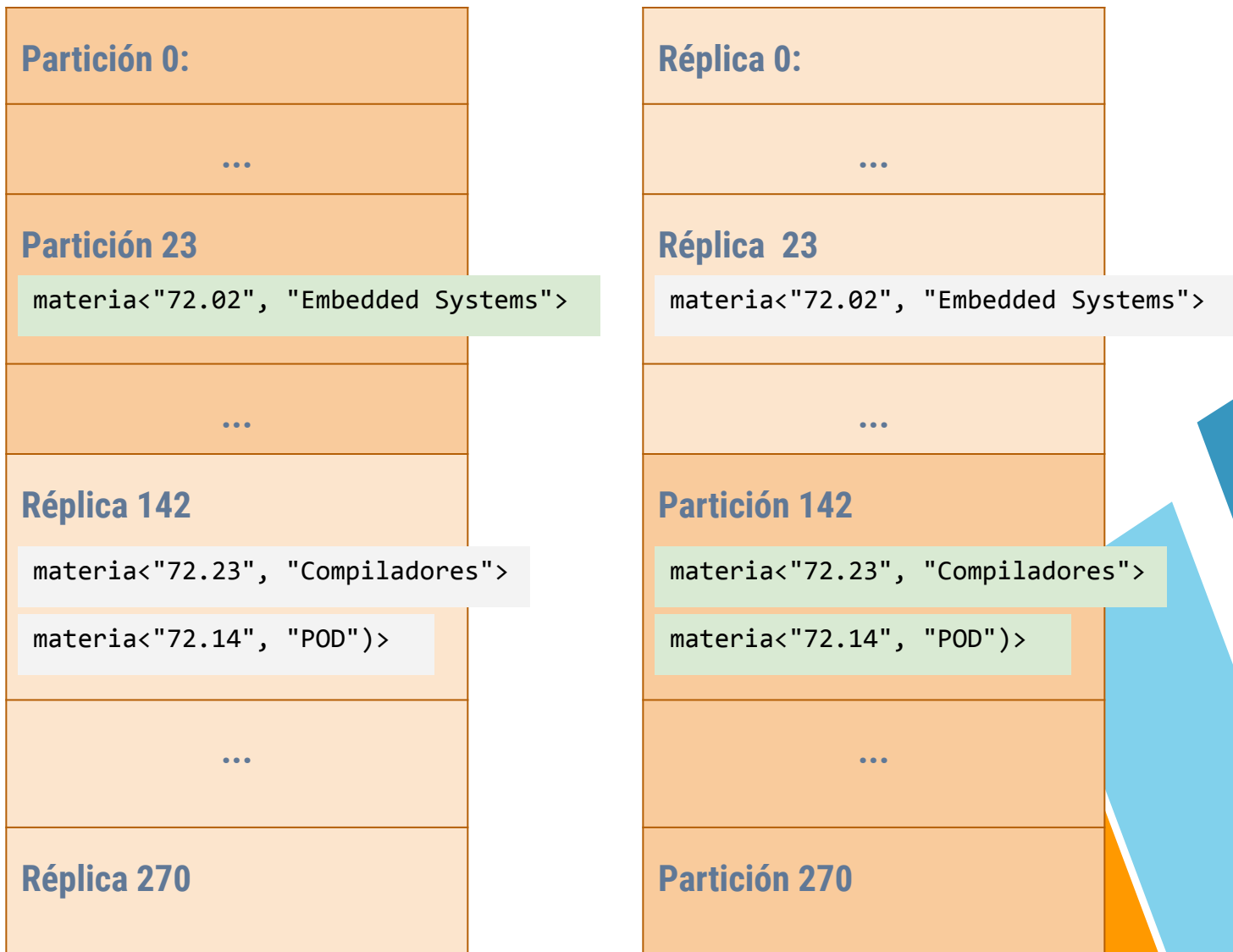
```
materia<"72.23", "Compiladores">
```

```
materia<"72.14", "POD">
```

...

Partición 270

# Hazelcast - Migración de Particiones






# Hazelcast - Réplicas Asincrónicas

En caso de querer que la escritura responda con mayor velocidad se puede configurar para que el **backup se haga de manera asincrónica**.

Para estas réplicas asincrónicas se garantiza “**consistencia eventual**”. Obviamente es más performante, pero puede haber problemas con los datos obtenidos, ya que se puede leer un dato *stale* (desactualizado).





# Hazelcast - Configuración



# Hazelcast- Configuración

Para poder configurar el cluster y que no se tomen valores por defecto hay **varias maneras** de configurarlos:

- » Declarativamente a través de un **archivo XML** (generalmente llamado hazelcast.xml)
- » Programáticamente desde el **cliente Java**
- » Utilizando **propiedades del sistema**
- » Usando **contexto de Spring**

Vamos a ver el modo programático en xml.





# Hazelcast - Member Discovery

Para que los miembros del cluster se encuentren hay varios modos configurables:

- » **Multicast**
- » **TCP/IP**
- » **Discovery con frameworks como Zookeeper, Consul**
- » **Discovery en clouds como AWS, AZURE**

```
//MULTICAST
<multicast enabled="true">
  <multicast-group>224.2.2.3</multicast-group>
  <multicast-port>54327</multicast-port>
  <multicast-time-to-live>32</multicast-time-to-live>
  <multicast-timeout-seconds>2</multicast-timeout-seconds>
  <trusted-interfaces>
    <interface>192.168.1.102</interface>
  </trusted-interfaces>
</multicast>
enabled="false"/>
<aws enabled="false"/>
```

```
//TCP IP
<multicast enabled="false" />
<tcp-ip enabled="true">
  <member>machine1</member>
  <member>machine2</member>
  <member>machine3:5799</member>
  <member>192.168.1.0-7</member>
  <member>192.168.1.21</member>
</tcp-ip>
```

El modo multicast no es recomendado en entornos productivos ya que comunicación vía UDP puede ser problemática y generalmente se bloquea.

# Hazelcast - Cluster Security

Aún con la misma configuración de *discovery* se pueden tener clusters independientes entre sí. Para ello dentro de la configuración se puede setear un "grupo" específico para cada cluster

En la misma se indica un nombre y una password para poder unirse al cluster, por lo que también provee una cierta "seguridad" al hecho de poder unirse al grupo.

```
<group>  
  <name>dev</name>  
  <password>dev-pass</password>  
</group>
```

La idea es que si un nodo de un cluster empieza a un nodo de otro cluster, se pasan el name y pass. Si no coincide, no se conectan.

# Hazelcast - Configuración de colecciones

Además de la configuración de red y clusters se puede configurar cada colección y otros elementos como el Executor Service.

```
<map name="default">
  <backup-count>1</backup-count>
  <async-backup-count>1</async-backup-count>
  <read-backup-data>false</read-backup-data>
</map>

<map name="materias">
  <backup-count>2</backup-count>
  <async-backup-count>1</async-backup-count>
  <read-backup-data>true</read-backup-data>
</map>
```

Ejemplo de config: quiero una replica sincronica, una replica asincronica, y que no se pueda leer de las replicas

En el ejemplo se configuran 2 mapas "**default**" y "**materias**".

- » Para **default** se crean 1 partición de backup sincrónica, 1 asincrónica y no se permite la lectura de las particiones de backup.
- » Para **materias** se configuran 2 backups sincrónicos y 1 asincrónicos y se puede leer de los mismos.

# Hazelcast - Split Brain

Dado un cluster se da un *split brain* cuando por problemas de comunicación el mismo se subdivide entre 2 partes autónomas.

Esto puede causar graves fallas, porque:

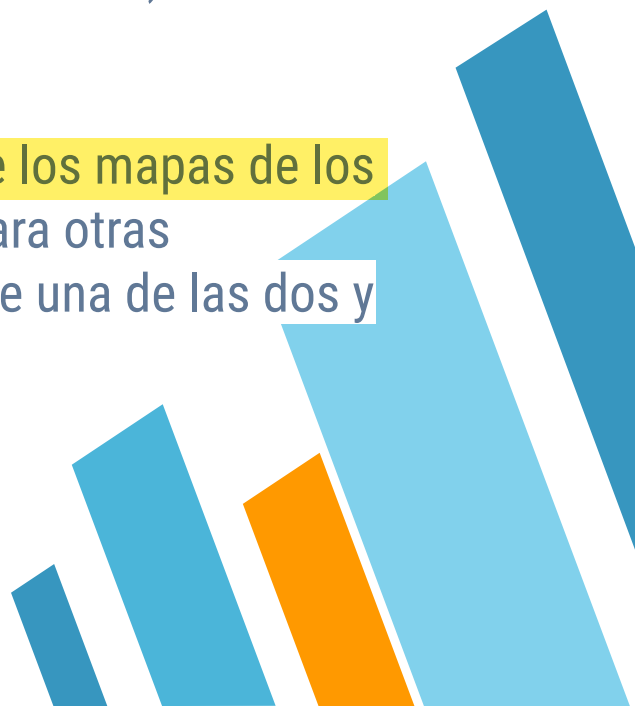
- Ambos clusters siguen funcionando
- Pueden recibir actualizaciones que conflictúan (por ejemplo ambos reciben 2 valores para la misma key)
- Si los "clusters" se vuelven a reconectar, cómo se realiza el merge de los datos

Ejemplo: tengo un cluster con 200 nodos. 100 en un país y 100 en otro. De repente se corta la comunicación entre esos dos países. Como Hazelcast es elastico, entonces los 100 de un lado eligen su propio master, y lo mismo con los otros 100. Los problemas surgen porque tengo dos bases distintas que siguen funcionando. Al momento de unirlos no se a quien le hago caso.



# Hazelcast - Split Brain

Para resolver un *split brain* en Hazelcast se realizan los siguientes pasos:

1. Elige un líder entre los dos líderes de los cluster (elige al más viejo).
  2. Se decide cuál de los dos clusters es el que debe mergearse (el más chico o por una función).
  3. Por cada miembro del cluster a mergear:
    - a. Pausar su operación (no acepta escrituras ni lecturas).
    - b. Cierra la conexión al cluster que pertenece.
    - c. Se une a otro cluster.
    - d. Envía un request para “mergear” los valores de los mapas de los cuales es dueño, según la política indicada. Para otras colecciones (listas, colas, etc) no mergea, elige una de las dos y se queda con esa
    - e. Reinicia su operación.
- 

# Hazelcast - Split Brain

```
<hazelcast>
...
<map name="default">
  <!--
    While recovering from split-brain (network partitioning),
    map entries in the small cluster will merge into the bigger cluster
    based on the policy set here. When an entry merge into the
    cluster, there might an existing entry with the same key already.
    Values of these entries might be different for that same key.
    Which value should be set for the key? Conflict is resolved by
    the policy set here. Default policy is hz.ADD_NEW_ENTRY
    There are built-in merge policies such as
    com.hazelcast.map.merge.PassThroughMergePolicy; entry will be added if
    there is no existing entry for the key.
    com.hazelcast.map.merge.PutIfAbsentMapMergePolicy ; entry will be
    added if the merging entry doesn't exist in the cluster.
    com.hazelcast.map.merge.HigherHitsMapMergePolicy ; entry with the
    higher hits wins.
    com.hazelcast.map.merge.LatestUpdateMapMergePolicy ; entry with the
    latest update wins.
  -->
  <merge-policy>MY_MERGE_POLICY_CLASS</merge-policy>
</map>
...
</hazelcast>
```

## Ejercicio 3 - Configuración.

1. Bajar de campus el archivo hazelcast.xml al CWD
2. Modificar el mismo para que
  - a. el *discovery* se haga por TCP y se encuentren sólo miembros en la computadora local
  - b. generar un grupo a partir del legajo de cada uno.
3. Correr la aplicación miembro en varias consolas
4. Comprobar que se conformó el cluster y se puede utilizar sin interferencia de otros

```
$> export CLASSPATH=$HOME/hazel/hazelcast-all-3.8.6.jar
```

```
$> java -Dhazelcast.config=$HOME/hazel/hazelcast.xml  
com.hazelcast.console.ConsoleApp
```



# **Hazelcast - Java**





# Hazelcast - Código Java

Para poder correr un nodo en Java se requiere:

- » Incorporar la dependencia (alcanza con el -cli, pero si tenemos el -all lo incluye)
- » Disponibilizar el xml de configuración (o configurar programáticamente).
- » Instanciar al objeto de Hazelcast mediante:

***HazelcastClient.newHazelcastClient(); //node client***

**o**

***Hazelcast.newHazelcastInstance(); //node member***

- » A partir de dicha instancia obtener la colección o elemento a utilizar y usarlo.
- 

# Hazelcast - Código Java

```
public class Cluster{  
    public static void main(String[] args)  {  
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();  
  
        Map<String, String> datos = hz.getMap("materias");  
  
        datos.put("72.42", "POD");  
  
        System.out.println(String.format("%d Datos en el cluster",  
                                          datos.size() ));  
  
        for (String key : datos.keySet()) {  
            System.out.println(String.format( "Datos con key %s= %s",  
                                              key, datos.get(key)));  
        }  
    }  
}
```

\$ java com.hazelcast.console.ConsoleApp

hazelcast[default] > ns MATERIAS

hazelcast[MATERIAS] > m .put 72.42 POD

hazelcast[MATERIAS] > m .size

hazelcast[MATERIAS] > m.iterator

## Ejercicio 4 - Código

1. Generar un proyecto Java con el arquetipo.
2. Incorporar la dependencia al jar de Hazelcast.
3. Incluir el hazelcast.xml para que quede en el CWD de la aplicación.
4. Copiar el código anterior
5. Comprobar el funcionamiento.



# **Hazelcast - Java Configuración**



# Hazelcast- Configuración via Java

Tanto server como client a la hora de instanciarse van a recibir un objeto de configuración al cual se le puede ir agregando mediante a setters diferentes tipos de configuraciones.

Estas configuraciones pueden ser:

- Seguridad
  - Conexión
  - Autenticación
  - Discovery
  - Red
  - Colecciones
  - etc.
- 

# Hazelcast - Server

Para instanciar un server necesitamos una instancia de Config y simplemente inicializar el server.

```
public class Server {  
  
    private static final Logger logger = LoggerFactory.getLogger(Server.class);  
  
    public static void main(String[] args) {  
        logger.info("hz-config Server Starting ...");  
  
        // Config  
        Config config = new Config();  
  
        ...  
  
        // Start cluster  
        Hazelcast.newHazelcastInstance(config);  
    }  
  
}}
```

# Hazelcast - Client

Para instanciar un client necesitamos un ClientConfig para instanciar la HazelcastClient que es el punto de entrada.


```
public class Client {  
  
    private static final Logger logger = LoggerFactory.getLogger(Client.class);  
  
    public static void main(String[] args) {  
        logger.info("hz-config Client Starting ...");  
  
        // Client Config  
        ClientConfig clientConfig = new ClientConfig();  
  
        HazelcastInstance hazelcastInstance =  
        HazelcastClient.newHazelcastClient(clientConfig);  
  
        // Shutdown  
        HazelcastClient.shutdownAll();  
    }  
}
```



# Hazelcast - Group Config

Tanto cliente como servidor pueden definir una configuración de grupo. Esta configuración sirve como una autenticación para que solo los nodos que pertenecen al mismo grupo formen un cluster. Y los que no pertenecen no pueden formar la configuración.

```
GroupConfig groupConfig = new GroupConfig()  
    .setName("cluster").setPassword("cluster-pass");  
  
config.setGroupConfig(groupConfig);
```







# Hazelcast - Network Config

Network configuration es la configuración que permite para los nodos del cluster saber como se conforma el cluster.

```
NetworkConfig networkConfig = new NetworkConfig();  
...  
config.setNetworkConfig(networkConfig);
```

# Hazelcast - (Network) Join Config

Join configuration es parte de la configuración de red e indica cómo los nodos del cluster van a poder encontrarse entre ellos.

```
MulticastConfig multicastConfig = new MulticastConfig();  
JoinConfig joinConfig = new JoinConfig().setMulticastConfig(multicastConfig);  
networkConfig.setJoin(joinConfig);
```

Alternativa listando los nodos del cluster (más seguro)

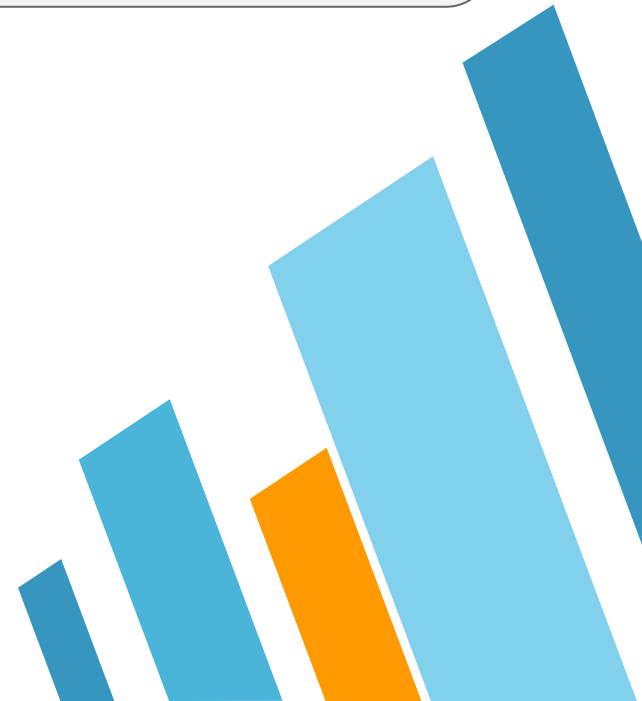
```
JoinConfig join = network.getJoin();  
join.getTcpIpConfig().addMember("10.45.67.32").addMember("10.45.67.100")  
    .setRequiredMember("192.168.10.100").setEnabled(true);
```



# Hazelcast - (Network) Interfaces Config

Permite configurar la interfaz de red que utilizar los nodos del cluster


```
InterfacesConfig interfacesConfig = new InterfacesConfig()  
.setInterfaces(Collections.singletonList("192.168.1.*"))  
.setEnabled(true);  
networkConfig.setInterfaces(interfacesConfig);
```





# Hazelcast - Network Otras configs

Otras configuraciones de red posibles pueden ser (no todas configurables por código)

- Public Address
  - Port
  - Outbound Ports
  - Reuse Address
  - Member Address Provider SPI
  - Advanced Network Configuration
- 

# Hazelcast - Client Config

La client config es la que le permite a un nodo client encontrar al cluster para realizar pedidos. Internamente maneja elementos homólogos (y algunos iguales) que el servidor

```
ClientConfig clientConfig = new ClientConfig();

// Group Config
GroupConfig groupConfig = new GroupConfig()
    .setName("cluster").setPassword("cluster-pass");
clientConfig.setGroupConfig(groupConfig);

// Client Network Config
ClientNetworkConfig clientNetworkConfig = new ClientNetworkConfig();
String[] addresses = {"192.168.1.51:5701"};
clientNetworkConfig.addAddress(addresses);
clientConfig.setNetworkConfig(clientNetworkConfig);
```



# Hazelcast - Management console

Hazelcast ofrece una webapp para monitorear en tiempo real el cluster. Nota: Es necesario utilizar una versión coincidente con la de hazelcast-all (Link).

```
// Management Center Config
ManagementCenterConfig managementCenterConfig = new
ManagementCenterConfig()
    .setUrl("http://localhost:32768/mancenter/")
    .setEnabled(true);
config.setManagementCenterConfig(managementCenterConfig);
```



# Hazelcast - Java Serialización



# Hazelcast - Serialización

Todas las clases que se quieran introducir en las colecciones deben distribuirse por los nodos vía red.

¿Entonces qué debe poder hacerse con las mismas?

**SERIALIZARSE**







# Hazelcast - Serialización

Para serializar, Hazelcast provee varios caminos:

- » Utilizar la interfaz `Serializable` de java
- » Utilizar la interfaz [`DataSerializable`](#) o [`IdentifiedDataSerializable`](#).
- » Utilizar interfaces alternativas como [`Portable`](#) o [`Externalizable`](#)
- » Escribir serializadores *custom*.

La recomendada es utilizar **`DataSerializable`** o **`IdentifiedDataSerializable`** ya que utiliza features de **Java NIO** para hacer más eficiente la serialización.



# Hazelcast - Serialización

```
public class Employee implements DataSerializable {
    //fields

    public Employee() {}
    //getters setters..

    public void writeData( ObjectDataOutput out ) throws IOException {
        out.writeUTF(firstName);
        out.writeUTF(lastName);
        out.writeInt(age);
        address.writeData (out);
    }

    public void readData( ObjectDataInput in ) throws IOException {
        firstName = in.readUTF();
        lastName = in.readUTF();
        age = in.readInt();
        address = new Address();
        address.readData(in); // since Address is DataSerializable let it
        read its own internal state

    }
}
```

Ejemplo de [DataSerializable](#)

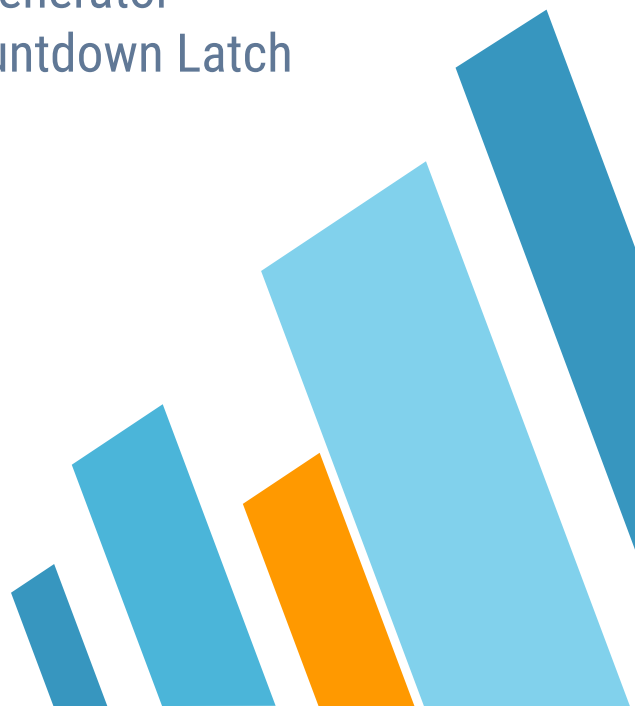


# **Hazelcast - Tipos de Elementos**



# Hazelcast - Tipos

Hazelcast define tipos que representan colecciones o elementos distribuidos:

- Map
  - MultiMap
  - Cache
  - Queue
  - Ringbuffer
  - Set
  - List
  - Replicated Map
  - Cardinality Estimator
  - Topic
  - Lock
  - Semaphore
  - AtomicLong
  - AtomicReference
  - IdGenerator
  - Countdown Latch
- 




# Hazelcast - Tipos Compatibilidad

Cada uno de estos tipos si tiene un equivalente en el lenguaje local intentan mantener la semántica.

En Java por ejemplo el mapa distribuido extiende del mapa de Java por lo que se lo puede usar como si “no fuera distribuido”

```
public interface IMap<K, V> extends ConcurrentMap<K, V>,  
LegacyAsyncMap<K, V> {
```



# Hazelcast - Mapa distribuido

```
public class DistributedMap {  
    public static void main(String[] args) {  
        Config config = new Config();  
        HazelcastInstance h = Hazelcast.newHazelcastInstance(config);  
        ConcurrentMap<String, String> map = h.getMap("my-distributed-map");  
  
        map.put("key", "value");  
        map.get("key");  
  
        //Concurrent Map methods  
        map.putIfAbsent("somekey", "somevalue");  
        map.replace("key", "value", "newvalue");  
    }  
}
```

# Hazelcast - Mapa distribuido con Trigger

```
public class EntryProcessorMain {
    public static void main(String[] args) {
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();
        IMap<String, Integer> map = hz.getMap("map");
        map.put("key", 0);
        map.executeOnKey("key", new IncEntryProcessor());
        System.out.println("new value:" + map.get("key"));
    }

    public static class IncEntryProcessor extends AbstractEntryProcessor<String,
Integer> {
        @Override
        public Object process(Map.Entry<String, Integer> entry) {
            int oldValue = entry.getValue();
            int newValue = oldValue + 1;
            entry.setValue(newValue);
            return null;
        }
    }
}
```

# Hazelcast - Map -> Query

```
public class Query {  
    public static void main(String[] args) {  
        Config config = new Config();  
        HazelcastInstance h = Hazelcast.newHazelcastInstance(config);  
        IMap<String, User> users = h.getMap("users");  
        // Just generate some random users  
        generateRandomUsers(users);  
  
        Predicate sqlQuery = new SqlPredicate("active AND age BETWEEN 18 AND 21");  
        Predicate criteriaQuery = Predicates.and(  
            Predicates.equal("active", true),  
            Predicates.between("age", 18, 21)  
        );  
  
        Collection<User> result1 = users.values(sqlQuery);  
        Collection<User> result2 = users.values(criteriaQuery);  
    }  
  
    public static class User implements Serializable {  
        String username;  
        int age;  
        boolean active;  
    }  
}
```



# Hazelcast - Queue


```
public class DistributedQueue {  
  
    public static void main(String[] args) throws InterruptedException {  
        Config config = new Config();  
        HazelcastInstance h = Hazelcast.newHazelcastInstance(config);  
        BlockingQueue<String> queue = h.getQueue("my-distributed-queue");  
        queue.offer("item");  
        String item = queue.poll();  
  
        //Timed blocking Operations  
        queue.offer("anotheritem", 500, TimeUnit.MILLISECONDS);  
        String anotherItem = queue.poll(5, TimeUnit.SECONDS);  
  
        //Indefinitely blocking Operations  
        queue.put("yetanotheritem");  
        String yetanother = queue.take();  
    }  
}
```

# Hazelcast - Topics

```
public class DistributedTopic implements MessageListener<String> {  
    public static void main(String[] args) {  
        Config config = new Config();  
        HazelcastInstance h = Hazelcast.newHazelcastInstance(config);  
        ITopic<String> topic = h.getTopic("my-distributed-topic");  
        topic.addMessageListener(new DistributedTopic());  
        topic.publish("Hello to distributed world");  
    }  
  
    @Override  
    public void onMessage(Message<String> message) {  
        System.out.println("Got message " +  
message.getMessageObject());  
    }  
}
```



# Referencias y para profundizar

- » <https://hazelcast.com>
  - » <https://docs.hazelcast.org/docs/3.8.6/manual/html-single/index.html>
  - » <https://docs.hazelcast.org/docs/3.8.6/manual/html-single/index.html#configuring-declaratively>
  - » <https://docs.hazelcast.org/docs/3.8.6/manual/html-single/index.html#serialization-interface-types>
  - » <https://docs.hazelcast.com/imdg/3.12/network-partitioning/network-partitioning>
  - » <https://www.ecyrd.com/cassandrascalculator/>
  - » <https://martin.kleppmann.com/2017/01/26/data-loss-in-large-clusters.html>
- 



# CREDITS

Content of the slides:

- » POD - ITBA

Images:

- » POD - ITBA
- » Or obtained from: [commons.wikimedia.org](https://commons.wikimedia.org)

Slides theme credit:

Special thanks to all the people who made and released these awesome resources for free:

- » Presentation template by [SlidesCarnival](https://slidescarnival.com)
  - » Photographs by [Unsplash](https://unsplash.com)
- 