Trabajo - Instalación, Administración y Uso de un servicio en Linux

Sistemas Grid: HazelCast - Avanzado

Servicios Telemáticos

Departamento de Ingeniería Telemática

Miguel Ángel Cabrera Miñagorri

ÍNDICE

1. Objetivos y Alcance

1.1. Introducción

En el presente trabajo se va a analizar y configurar un clúster de servidores HazelCast. HazelCast es un In-Memory Data Grid, en adelante IMDG. Este tipo de servicios surgen de la necesidad de tener una gran cantidad de datos accesibles en memoria RAM, del orden de TeraBytes, dada la mayor velocidad que esta presenta frente a los datos almacenados en el disco. Además, un IMDG nos permite que este almacenamiento de datos sea distribuido de forma que varias máquinas pueden acceder a los datos sin necesidad de que estén almacenados en ellas. Aunque los IMDG se pueden utilizar como simples bases de datos distribuidas este no es su verdadero propósito y por ello no son la tecnología más adecuada, su gran aporte práctico es la computación distribuida, esto es, la resolución de problemas complejos por medio de varias máquinas que cooperan entre sí y comparten sus datos para lograr un objetivo común, siendo especialmente útiles en soluciones Big Data.

En este punto cabe destacar la diferencia entre un IMDG y un In-Memory Compute Grid, mientras que en el primero lo que se intercambia son datos, en el segundo lo que se intercambia son tareas, es decir, el clúster está formado por nodos que se reparten las tareas a realizar (tareas que el clúster recibe de los clientes), mientras que en el IMDG se reparten los datos (datos que se reciben de los clientes, **clientes que normalmente son servidores de otras aplicaciones que necesitan compartir datos para ser escalables**), aunque el propio equipo que forma parte del clúster puede realizar tareas y guardar en el clúster los datos obtenidos. En cuanto a esto, HazelCast IMDG es un nombre un poco confuso pues además de ser un IMDG permite ciertas funcionalidades de realización de tareas distribuidas y ciertas características de computación.

1.2. Motivación y funcionalidad del servicio

Como se ha comentado anteriormente, lo que puede llevar al uso de un IMDG es necesitar manipular grandes cantidades de datos de forma que se encuentren altamente accesibles. Esta es una afirmación muy general, por lo que los ejemplos de funcionalidad expuestos a continuación pretenden concretar que tipo de usos son posibles, y los más adecuados.

En cuanto a la funcionalidad de los IMDG, existe un abanico muy amplio de casos de uso ya que se trata de una tecnología que da solución a un problema recurrente en varios

campos. A continuación, se citarán algunos de estos casos de uso posibles, que además de ser genéricos de este tipo de servicios, son también usos frecuentes y especializaciones del servicio estudiado, como puede comprobarse en [1].

- Además de la computación distribuida, comentada en la introducción, los IMDG permiten distribuir estructuras de datos, como puede ser un mapa (clave/valor), una cola (Queue), una lista (List), etc. Un ejemplo concreto de esto en el que se profundizará más adelante en los escenarios es el almacenamiento de sesiones web de forma distribuida, lo que permite que un usuario recupere la sesión, de forma totalmente transparente, contra otro servidor cuando el primero al que estaba conectado cae.

- Otra de sus grandes utilidades es el escalado automático de aplicaciones, por ejemplo, en aplicaciones con una estructura cliente-servidor, si el número de clientes se dispara de repente nuevos servidores se añadirían automáticamente al clúster permitiendo que dichos clientes fuesen atendidos sin problema, y si es necesario mover clientes a otro servidor sin pérdida de datos.

- También, pueden ser usados como "backbone" para una estructura para el despliegue de microservicios.

En general, como se ha comentado, las posibilidades de uso son varias, pero actualmente la mayor implicación es en la computación distribuida, en la que también se profundizará más adelante en los escenarios.

1.3. Documentación bibliográfica

Se muestra a continuación una lista de referencias bibliográficas que han sido de ayuda en la comprensión del servicio:

[1] https://hazelcast.org/use-cases/messaging/

[2] https://www.gridgain.com/resources/blog/in-memory-data-grid-explained

[3] https://hazelcast.org/features/#DistributedQuery

[4] https://hazelcast.org/documentation/#open-binary

[5] https://blog.hazelcast.com/hazelcast-vs-elasticache-memcached/

[6] https://docs.hazelcast.org/docs/3.9/manual/html-single/index.html#discovery-mechanisms

[7] https://docs.hazelcast.org/docs/3.9/manual/html-single/index.html#how-distributed-query-works

[8] https://hazelcast.org/use-cases/data-grid/

[9] https://hazelcast.org/use-cases/caching/

[10] https://hazelcast.org/use-cases/in-memory-nosql/

[11] https://hazelcast.org/use-cases/application-scaling/

[12] https://hazelcast.org/use-cases/clustering/

[13] https://hazelcast.org/use-cases/microservices/

[14] https://hazelcast.org/use-cases/messaging/

[15] https://hazelcast.org/features/#DistributedDataStructures

[16] https://hazelcast.org/features/#DistributedCompute

[17] https://hazelcast.org/features/

[18] https://www.gridgain.com

[19] http://java.candidjava.com/tutorial/Servlet-Jsp-HttpSession-Login-logout-example.html

[20] https://hazelcast.org

[21] https://github.com/hazelcast/hazelcast

[22] https://maven.apache.org/

[23] http://cr.openjdk.java.net/~ihse/demo-new-build-readme/common/doc/building.html

[24] https://hazelcast.org/download/#imdg

[25] https://github.com/hazelcast/hazelcast/releases/tag/v3.10.6

[26] https://docs.hazelcast.org/docs/3.0/manual/html/ch12.html

[27] https://docs.hazelcast.org/docs/latest/manual/html-single/#using-the-script-cluster-sh

[28] https://docs.hazelcast.org/docs/latest/manual/html-single/#health-check-script

[29] https://docs.hazelcast.org/docs/latest/manual/html-single/#diagnostics-plugins

[30] https://docs.hazelcast.org/docs/latest/manual/html-single/#security

[31] https://docs.hazelcast.org/docs/2.0/manual/html/ch09.html

[32] https://github.com/hazelcast/hazelcast/blob/master/hazelcast/src/main/resources/hazelcast-full-example.xml

2. Base Teórica

A continuación, se van a exponer una serie de conceptos e ideas en las que se basan los servicios IMDG y que son necesarias para comprender el funcionamiento del servicio estudiado.

Los IMDG organizan los nodos (máquinas) en clústeres. Un clúster es una agrupación de máquinas (nodos o miembros) que se reparten datos entre ellas. Los datos se dividen en particiones y cada nodo del clúster almacena un cierto número de particiones. No perder de vista que estas particiones se almacenan en memoria RAM, lo que aporta una mayor velocidad en la obtención de los datos. Además del reparto de las particiones principales se realiza un reparto de particiones de back-up, de forma que un nodo mantiene un número de particiones principales y un número de particiones de back-up que corresponden a particiones principales almacenadas en otros nodos. Estas particiones de back-up se utilizan cuando un nodo cae, al caer el nodo los datos que mantenía en su RAM se pierden, en este momento se utilizan las particiones de back-up y se vuelven a repartir las particiones principales entre todos los nodos. Por ello la pérdida total de los datos solo puede producirse en el caso de que no exista ningún nodo en el clúster, es decir, deje de existir el clúster.

Los sistemas IMDG suelen utilizar autodescubrimiento de nodos para formar el clúster de manera totalmente automática. Esta operación puede realizarse de diversas formas, por ejemplo, mediante el uso de direcciones ip multicast, estableciendo en la configuración los las direcciones ip de cada nodo, etc.

Visto desde muy alto nivel, tal como se explica en [2], un IMDG podría verse como un gran mapa clave/valor, en el que se introducen unos datos con una clave que posteriormente se utiliza para obtenerlos, como un típico HashMap. Con la peculiaridad de que tanto la clave como el valor podría ser cualquier tipo de objeto, no estaría limitado a un "array" o "string" como en los sistemas tradicionales, y es esta una de las principales diferencias entre un IMDG y un IMDB (In-Memory Data Base).

Para obtener datos de un clúster existiría la posibilidad de iterar sobre las claves de cada nodo hasta encontrar la coincidente, pero esto supondría pedir a cada nodo del clúster el set de claves/valores e iterar localmente sobre ellas. Para solucionar este problema existe lo denominado "Distributed Query" que se encarga de realizar una consulta distribuida en el clúster sobre la estructura de datos que se compone entre todos los nodos y devolver exclusivamente el dato pedido. De esta forma, la consulta se envía a todos los nodos del clúster, cada nodo realiza la búsqueda entre sus entradas de datos y por último se fusiona la respuesta de cada nodo antes de enviarla de vuelta al cliente. Puede obtenerse más información acerca de este proceso en [7].

Atendiendo a lo expuesto, el proceso de inserción u obtención de datos sería el siguiente:

1. El cliente se conecta al clúster, esta conexión podría ser directamente a uno de los nodos o podría pasarse por un balanceador de carga que decide el nodo al que debe ser conectado el cliente.

2. El cliente puede solicitar la inserción de nuevos datos, lo que supone una distribución de estos mediante el mecanismo de particiones, o puede solicitar la obtención de datos, lo que supone la realización de la consulta distribuida ("Distributed Query").

Con esta breve explicación sobre el funcionamiento de un IMDG genérico se pasará a comentar los aspectos concretos de HazelCast, profundizando así en este servicio.

2.1. Descripción del servicio y conceptos implicados

HazelCast como IMDG emplea todos los conceptos teóricos expuestos anteriormente. Por ello se va a pasar a concretar como funciona en HazelCast cada uno de ellos.

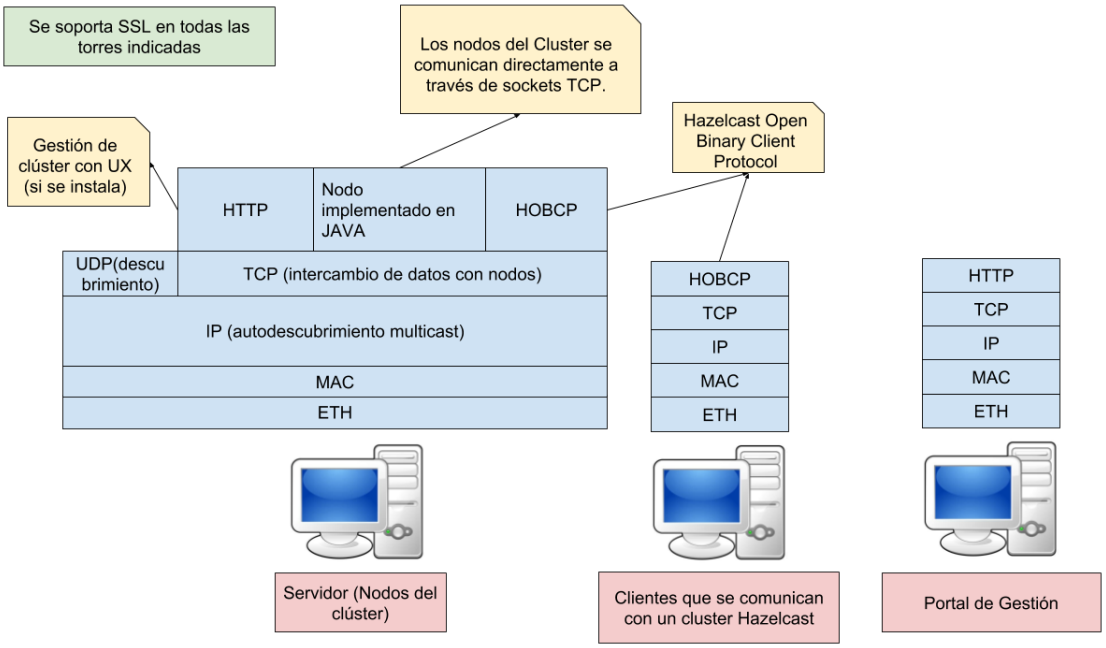
Hazelcast nos ofrece dos formas genéricas de realizar el autodescubrimiento que son el uso de ip multicast, donde habría que configurar la dirección del grupo multicast, esta opción se recomienda cuando los nodos están en la misma LAN o directamente mediante tcp/ip, que sería útil para el caso en que los nodos fuesen conocidos puesto que hay que configurar la dirección ip de cada nodo. Además de estas dos formas genéricas, HazelCast nos ofrece un protocolo de autodescubrimiento para el despliegue en AWS (Amazon Web Services), Microsoft Azure, Heroku, etc. Es posible observar todas las posibilidades en [6].

En cuanto a la distribución de particiones, en HazelCast existen 271 particiones principales en el clúster, y por cada una de ellas otra de back-up. El reparto entre nodos es equitativo, es decir, cada nodo almacena 271÷(número de nodos) particiones. Y lo mismo ocurre con las particiones de back-up. Aunque el número de particiones puede ser configurado. Además, existe una característica peculiar y es que pueden asignarse particiones a las instancias que corren sobre la misma JVM, es decir, un numero n de particiones para todas las instancias HazelCast que corren en la misma JVM en vez de individualmente por cada una.

Respecto a la "Distributed Query" en HazelCast existen cuatro tipos diferentes, en los que no se profundizará puesto que no han sido tenidos en cuenta en la parte práctica del trabajo. En [3] puede obtenerse información acerca de ellos.

2.2. Protocolos utilizados por el servicio

A continuación, se van a presentar los protocolos utilizados por HazelCast. Se presenta primero una imagen con las torres de protocolos utilizadas por cada entidad.



Para comprender algunos aspectos de la imagen anterior es importante conocer que no existe un cliente propio de HazelCast, sino que HazelCast proporciona una API para varios lenguajes y cada usuario crea su cliente en función de sus necesidades, normalmente se integra el código para la comunicación con el clúster en el código de la aplicación que se esté desarrollando. Por ello, al escribir código para la comunicación con un clúster se hace uso indirectamente del protocolo HOBCP (Hazelcast Open Binary Client Protocol) a través de la API. Además, cuando se usan este tipo de servicios el cliente HazelCast suele ser el servidor de la aplicación que necesita utilizarlo.

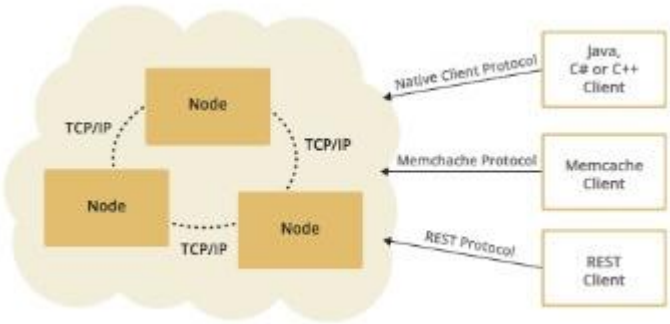
Por otro lado, el portal de gestión se comunica con los nodos a través de una API REST, y este es una aplicación web basada en ".war" que permite la gestión del clúster a través de un navegador web.

Destacar que, aunque no aparezca en la pila, es posible el uso de SSL en todas las entidades que aparecen en la imagen.

El protocolo UDP se utiliza entre nodos para el autodescubrimiento, en el caso de que se haya configurado con ip multicast. En otro caso se realiza mediante TCP contra las direcciones ip unicast configuradas. O puede configurarse el mecanismo de autodescubrimiento para entornos específicos como AWS, Azure, etc.

Para la comunicación entre nodos del clúster se hace uso directamente de sockets TCP, no existiendo un protocolo que la defina explícitamente.

A continuación, se muestra una imagen sacada de la documentación, que pone de manifiesto los protocolos utilizados por los clientes y entre nodos. En ella, cuando hace referencia a "Native Client Protocol" se refiere a HOBCP. El protocolo MemCache no es un protocolo específico de HazelCast, es de Memcached, una empresa dedicada a la cache distribuida. HazelCast permite compatibilidad con este, es decir, una aplicación que utilice el protocolo MemCache puede hablar con un clúster HazelCast. Esta compatibilidad hay que configurarla y no está activada por defecto. Al ser su uso excepcional, no ser propio de HazelCast y no haber sido usado en la realización del trabajo no nos centraremos en este. La imagen ha sido obtenida de una comparación entre MemCached y HazelCast que puede leerse completa en [5]. Y por último, el cliente REST, que se usa principalmente en el portal de gestión para gestionar (dar órdenes a) los clústeres.



2.2.1. Protocolos Comunes

Como protocolos comunes tenemos:

- HTTP: definida la versión HTTP/1.1 en la RFC2616.
- SSL: definida la versión 3.0 en la RFC6101.

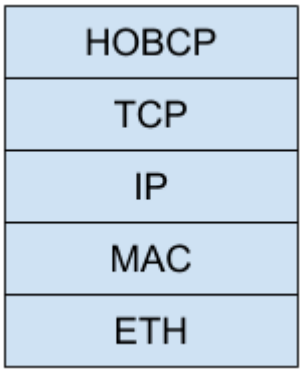
2.2.2. Protocolos Específicos del servicio

Para la comunicación entre clientes y el clúster, se hace uso del protocolo HOBCP indirectamente a través de la API de HazelCast. Se puede descargar la definición de este protocolo en formato pdf desde [4]. Este protocolo se utiliza tanto para el intercambio de

datos como para la autenticación del emisor. Brevemente descrito, el protocolo funciona de la siguiente manera:

1. El cliente envía al servidor un mensaje de apertura con “CB2”, estos 3 caracteres indican la versión y “CB2” es la única version activa del protocolo actualmente.
2. El servidor asiente la apertura de la connexion.
3. El cliente envía un mensaje de autenticación con la credenciales necesarias para autenticarse.
4. El cliente puede enviar datos, mensajes de suscripción a eventos, etc.
5. En caso de que no haya datos que enviar se envían mensajes de refresco para mantener abierta la connexion.
6. Por ultimo, se cierra el socket y el servidor libera los recursos asociados a la conexión.

Se presenta ahora la pila de protocolos que da soporte a HOBCP:



En cuanto a los mensajes utilizados por HOBCP, el mensaje enviado por el cliente siempre tendrá una cabecera de longitud fija y una carga de datos de longitud variable, se muestra esquema a continuación:

MESSAGE HEADER		MESSAGE HEADER EXTENSION	MESSAGE BODY
20 BYTES	data-offset field, 2 bytes	0 to (2^16-22) bytes Currently Unused	PAYLOAD
<----------DATA OFFSET DEFINES THIS SIZE--------->			

Donde la cabecera consta de los siguientes campos:

Name	Data Type	Description
Frame Size	int32	Frame data size is the total size in bytes. The minimum value of the frame size can be a fixed header size of 22 bytes: header message only, no payload.
Version	uint8	Protocol version. Current version is 1.
Flags	uint8	Flag bits. Please see the table below for an explanation of the flags.
Type	uint16	Type of the message corresponding to a unique operation of a distributed object: e.g. map.put, map.get, etc., a response, a general protocol message type, or an event.
Correlation ID	int64	This ID correlates the requests to responses. It should be unique to identify one message in the communication. This ID is used to track the request response cycle of a client operation. Server side should send response messages with the same ID as the request message. The uniqueness is per connection. Each message generating entity must generate unique IDs. If the client receives the response to a request and the request is not a multi-response request (i.e. not a request for event transmission), then the correlation ID for the request can be reused by subsequent requests.
Partition ID	int32	Target partition where this message will be processed. Negative values will go into a global execution pool and do not have a partition.
Data Offset	uint16	A message may have a payload data. Data offset value must be set to the number of bytes from the beginning of the frame to the start of the payload including the "Data Offset" field.

Y el campo de flags tiene la siguiente estructura:

8	7	6	5	4	3	2	1
BEGIN	END	N/A	N/A	N/A	N/A	N/A	EVENT

Los flags de BEGIN y END se utilizan para enviar datos fragmentados en el body, y los datos del body empiezan donde indique el campo “data-offset”.

Cabe destacar que la cabecera puede ser extendida por el propio desarrollador del cliente en el campo “MESSAGE HEADER EXTENSION”.

Dado que el protocol tiene una gran cantidad de mensajes se indicará otro más, en este caso el mensaje de error que envía el servidor al cliente en caso de error.

Error Message

Response Message Type Id: 109

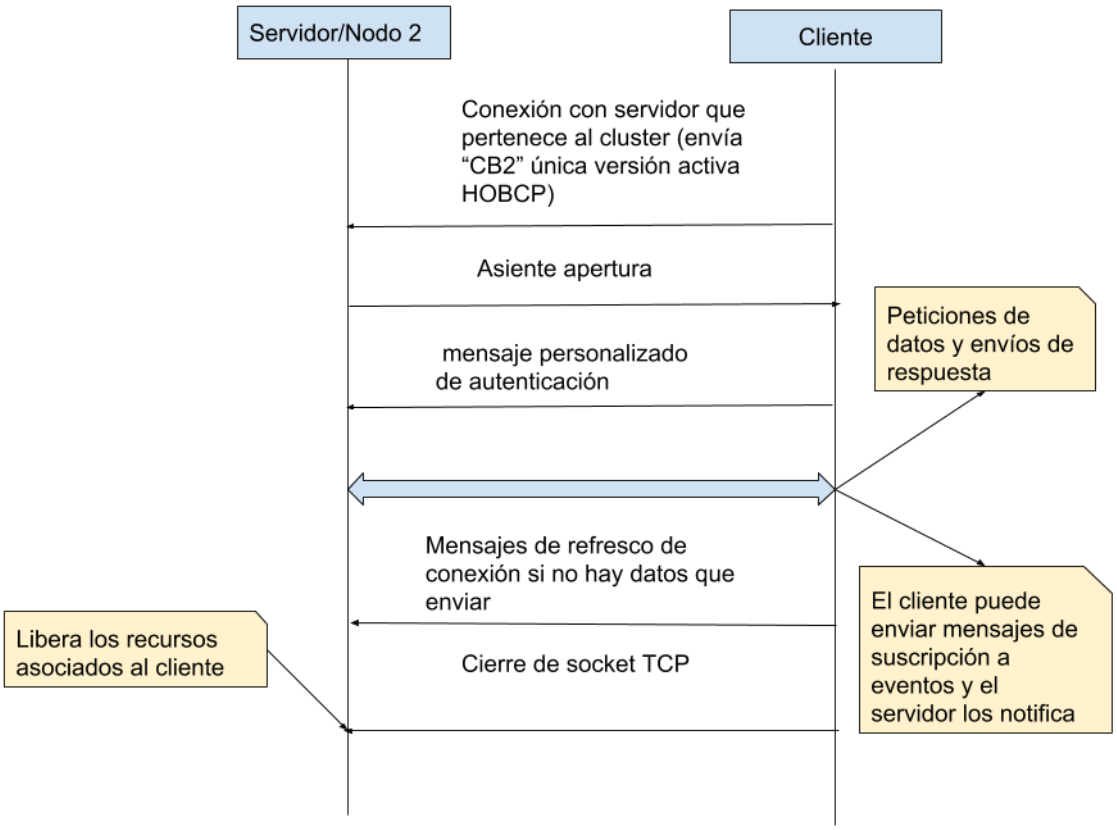
Field	Type	Nullable	Description
Error Code	int32	No	The unique code identifying the error
Class Name	string	No	The class name which caused the error at the server side
Message	string	Yes	The brief description of the error
Stack Trace	array of stack-trace	No	The stack trace at the server side when the error occurred
Cause Error Code	int32	No	The error code for the actual cause of the error. If no cause exists, it is set to -1
Cause Class Name	string	Yes	The name of the class that actually cause the error

Dado el tamaño de la tabla que indica el significado de cada código de error no se incluirá. Puede verse consultando [4] y descargando el pdf asociado al protocolo.

Por ultimo, se incluirá el mensaje de autenticación personalizada que el cliente debe enviar siempre al servidor como mensaje posterior a la apertura de la conexión. Si no se envía mensaje de autenticación no podrá realizarse ninguna operación. Existen varios mensajes de autenticación, por lo que se mostrará unicamente el mensaje personalizable de autenticación, notar que estos serían los campos que irían en el payload del mensaje genérico expuesto anteriormente:

Name	Type	Nullable	Description	Available since
credentials	byte-array	No	Secret byte array for authentication.	1.0
uuid	string	Yes	Unique string identifying the connected client uniquely. This string is generated by the owner member server on initial connection. When the client connects to a non-owner member it sets this field on the request.	1.0
ownerUuid	string	Yes	Unique string identifying the server member uniquely.	1.0
isOwnerConnection	boolean	No	You must set this field to true while connecting to the owner member, otherwise set to false.	1.0
clientType	string	No	The type of the client. E.g. JAVA, CPP, CSHARP, etc.	1.0
serializationVersion	uint8	No	client side supported version to inform server side	1.0

Para finalizar este apartado se mostrará un diagrama de secuencia del protocolo HOBCP:



En cuanto a la comunicación entre los nodos del cluster, es llevada a cabo directamente sobre sockets TCP, como un sistema de comunicación P2P (Peer to peer), no existiendo un protocolo definido para ello. No ha sido posible encontrar información relativa a como se estructuran dichos mensajes, ya que ni en la propia documentación de HazelCast hace mención de ello.

3. Evaluación de la implementación estudiada

A continuación, se realizará un análisis comparativo entre HazelCast y otros IMDG existentes en el mercado. Se adelanta que uno de los puntos más fuertes de HazelCast es el ser **OpenSource**, aunque no siendo el único.

3.1. Descripción y características de la implementación de servicio estudiada

HazelCast es una solución altamente utilizada por muchas empresas del sector, soporta gran cantidad de lenguajes de programación y ofrece varias funcionalidades con poca configuración. En los siguientes puntos se realizará una descripción detallada del mismo.

3.1.1. Breve Descripción de la solución adoptada

Como se ha comentado HazelCast es un IMDG OpenSource, que ofrece librerías para varios lenguajes de programación mediante las cuales es posible la creación de clientes que se comuniquen con el clúster, incluso la programación del comportamiento de un miembro del clúster. Además, ofrece facilidades con una configuración sencilla para ciertos problemas recurrentes que pueden solucionarse con un IMDG, en concreto problemas para los que HazelCast es eficiente, que se muestran en la tabla siguiente, aunque puede ser utilizado con otros fines:

Funcionalidad	**Versión Usada**	**Ultima versión**	**Autor**	**Referencias**
IMDG (uso de un IMDG puro tal como se describió al principio de este trabajo)	3.11 23-10-2018	3.11 23-10-2018	HazelCast	[8]
Caching (Soluciones de cacheo de datos para garantizar que los datos están disponibles donde se necesitan)	3.11 23-10-2018	3.11 23-10-2018	HazelCast	[9]
In-Memory NoSQL (Solución de base de datos no relacional en memoria distribuida)	3.11 23-10-2018	3.11 23-10-2018	HazelCast	[10]
Application Scaling (Solución para el escalado de aplicaciones que ofrecen servicios)	3.11 23-10-2018	3.11 23-10-2018	HazelCast	[11]

Web Session Clustering (Almacenamiento de sesiones web en el clúster para evitar la pérdida de estas al caer nodos servidores)	3.11 23-10-2018	3.11 23-10-2018	HazelCast	[12]
Microservices (Puede ser usado como "backbone" o núcleo de una arquitectura de microservicios)	3.11 23-10-2018	3.11 23-10-2018	HazelCast	[13]
Messaging (Puede ser utilizado como sistema de difusión de mensajes, por ejemplo, para publicar eventos en un bus con varios receptores)	3.11 23-10-2018	3.11 23-10-2018	HazelCast	[14]

Nota: La versión de HazelCast es la misma sea cual sea el uso que se le da, esto es debido a que se descarga el servidor HazelCast completo y no se distingue por funcionalidad.

3.1.2. Equipamiento necesario

A continuación, se muestra una tabla donde se indica el equipamiento necesario para soportar el servicio:

Hardware	**Infraestructura de red**	**Software**
Mínimo un equipo donde se pueden instanciar varios nodos.	Mínimo 2 nodos, para garantizar que no hay pérdida de datos en la caída de alguno.	JVM, cualquier versión.
-	-	Ninguna dependencia adicional.

-	-	Cualquier SO que soporte la JVM. **Se usará CentOS 7**.
-	-	Un cliente escrito en: java, c#, c++, node.js, python o go
-	-	El propio servidor de HazelCast.

Además, en el trabajo se hará uso de un servidor Apache y un servidor Tomcat en uno de los escenarios para demostrar el funcionamento del Web Session Clustering, aunque no aparecen en la tabla anterior porque no son un requisito para que HazelCast funcione.

3.1.3. Características y funcionalidades

Además de las funcionalidades que se han mostrado en la tabla del *apartado 3.1.1* HazelCast ofrece las siguientes características:

Característica	Descripción
Distribución de estructuras de datos	Hazelcast ofrece una gran cantidad de estructuras de datos para manipular de forma distribuida como pueden ser mapas, colas, listas, semáforos, etc. Pueden verse todas en [15].
Distribución de la computación	Como se comentó, Hazelcast no es un simple IMDG, tambien ofrece varias formas de distribuir la computación en el clúster. Haciendo uso de Executor Service de java, Entry Procesor, etc. Consultar [16]
Consulta distribuida o "Distributed Query"	Hazelcast ofrece varias maneras de realizar la consulta distribuida.
Integrated Clustering	Hazelcast permite realizar el clustering de datos de forma integrada con algunos servidores basados en java como por

	ejemplo Tomcat, lo que facilita la configuración.
Standars	HazelCast utiliza para el clustering de datos los standars JCache y Apache Jclouds Support.
Soporte para Cloud y virtualización	HazelCast permite el despliegue en múltiples entornos Cloud como AWS, Azure, Heroku, etc.
Gestión del Clúster	HazelCast hace uso de JMX y obtiene estadísticas, ambos por cada nodo.
Big Data	Hazelcast ofrece algunas funcionalidades para BigData como HazelCast Jet, un conector para Apache Spark y soporta la integración con Mesos.

Toda la información de la tabla anterior puede ampliarse en [17].

3.2. Comparativa de soluciones existentes en el mercado

A continuación, se muestra una comparativa de algunas de las implementaciones existentes en el mercado que hacen competencia al servicio estudiado:

Parámetro	**HazelCast**	**Oracle Coherence**	**GridGain**	**Apache Ignite**
S.O. soportados	Cualquiera con JVM.	Cualquiera con JVM.	Linux OS X Solaris Windows	Cualquiera con JVM.
Equipo de	HazelCast	Oracle, aunque	GridGain	Apache

desarrollo		fue comprado a Tangosol.	Systems, pero construido sobre el proyecto de código abierto Apache Ignite, del que son un gran contribuidor.	Software Foundation
Licencia	Apache License 2.0 (OpenSource), existe otra de uso comercial.	Comercial	Comercial	Apache License 2.0 (Open Source)
Actualmente	En progreso.	En progreso.	En progreso.	En progreso.
Pago/Gratuito	Una versión de pago y otra gratuita.	Pago.	Pago.	Gratuito.
Dirección de distribución	https://hazelcast.org/download/#imdg		https://www.gridgain.com/products/software/professional-edition	
Lenguaje en el que está desarrollado	Java	Java	Java, C++, .Net	Java mayormente, C++, Python, .NET
Lenguajes soportados por los clientes	Nativamente: .NET, C++, Python, node.js,	Java, C++, .Net También soporta	Java, C++, .Net También	Nativamente: .Net, C++, java. Con pluggins:

	java, Scala, Go. Lenguajes disponibles según versión. La última versión no los soporta todos. También soporta clientes REST. Compatible con clientes Memcached	clientes REST.	soporta clientes REST.	Python, php, node.js También soporta clientes REST. Compatible con clientes Memcached
Nivel de uso	16%	7%	14%	4%
Dependencias	JVM.	JVM.	JVM, Apache Ignite	JVM.
Fecha última versión	30 Octubre 2018	Abril 2018	11 Diciembre 2018	16 Julio 2018
Comentarios	IMDG es una de sus múltiples características.		IMDG es una de sus múltiples características.	No es exactamente un IMDG, ellos mismos lo definen como una plataforma para proporcionar una base de datos distribuida, caching y procesamiento de grandes cantidades de

				datos.

Los datos de nivel de uso han sido estimados por mí a partir de los clientes que cada opción tiene y los beneficios que obtienen. No son datos totalmente reales. El resto de porcentaje que falta hasta el 100% se debe a otras soluciones del mercado. En cualquier caso, HazelCast y GridGain son los más usados de los cuatro competidores que se han presentado. Como dato, Apache Ignite es tomado por varias empresas como base para crear su solución, como por ejemplo ha hecho GridGain.

Queda constancia de que el principal competidor de HazelCast es GridGain, de hecho, en el sitio web de GridGain [18] se comparan directamente con HazelCast. Pero, a pesar de algunas diferencias de funcionalidad, HazelCast, por lo que he podido observar al realizar las búsquedas para encontrar los datos de la tabla anterior, tiene una documentación más amplia y de mayor calidad que GridGain, lo cual puede ser el motivo de su superioridad de mercado.

En cuanto a funcionalidad, GridGain tiene algunas características que no tiene Hazelcast como por ejemplo In-Memory File System, soporte para MongoDB, … Sin embargo, también existen funcionalidades que posee HazelCast y no GridGain como por ejemplo en Hazelcast el esquema de datos es libre (se puede almacenar cualquier tipo de objeto sin necesidad de conversiones), mientras que en GridGain no. HazelCast es compatible con clientes MemCached mientras que GridGain no. En definitiva, cada uno tiene unas funcionalidades por las que se diferencia del otro, no siendo ni mejor ni peor, simplemente hay que buscar el que más se adecue al caso de uso.

Respecto a Oracle Coherence y Apache Ignite, HazelCast soporta de forma nativa bastantes más lenguajes.

Además, HazelCast permite un despliegue sencillo en muchos entornos Cloud y ha creado sus mecanismos de autodescubrimiento de nodos especializados para ellos. Actualmente el uso de este tipo de entornos está en auge por lo que HazelCast parece la mejor opción si se desea utilizar estos entornos.

Nota aclarativa: algunos campos de la tabla anterior como por ejemplo la licencia de Apache Ignite no concuerdan con el de la presentación debido a que se había interpretado mal la licencia Apache License 2.0 y no se había comprendido que es una licencia de código abierto.

3.3. Clientes para el servicio

No procede debido a que no existe un cliente en sí. Como se ha comentado antes cada usuario crea su propia aplicación cliente a partir de la API en el lenguaje que desee siempre que este esté soportado, por ello, para la realización del trabajo se hará uso de clientes de ejemplo que se obtienen de la página de HazelCast y una aplicación web que hace uso de sesiones (HttpSession) que se ha descargado en ".war" de [19] para hacer uso del Web Session Clustering.

3.3.1. Referencias y características del cliente adoptado

No procede. Idem apartado anterior.

3.3.2. Comparativa de clientes existentes en el mercado

No procede. Idem apartado anterior.

4. Proceso de instalación y Uso del cliente

No procede debido a que no existe un cliente en sí. Para utilizar un cliente simplemente hay que hacer uso de la API en el lenguaje deseado y la API se encarga de ejecutar el protocolo HOBCP para comunicarse con el clúster. Para la realización del trabajo se hará uso de aplicaciones clientes de prueba descargados de la página oficial de Hazelcast.

4.1. Obtención del software del cliente

No procede. Idem apartado anterior.

4.2. Instalación del cliente

No procede. Idem apartado anterior.

4.2.1. Primera instalación del cliente

No procede. Idem apartado anterior.

4.2.2. Desinstalación del cliente

No procede. Idem apartado anterior.

4.3. Configuración del cliente

No procede. Idem apartado anterior.

5. Proceso de instalación/administración del servidor

Se va a pasar ahora a explicar los pasos que se deben seguir para la obtención, instalación y configuración del servidor HazelCast.

5.1. Obtención del software del servidor

La obtención del software del servidor puede realizarse de dos formas. La primera, y la más simple, descargar el .zip o .tar de la página oficial [20]. Una vez descargado y descomprimido, al estar escrito en java, se obtendrá un archivo .jar que es el ejecutable del servidor. La segunda forma es clonar el repositorio de código desde GitHub en [21] (está disponible puesto que es de código abierto) con el comando `git clone https://github.com/hazelcast/hazelcast.git` y crear el bytecode mediante el comando `mvn clean install`, este comando se encargará de compilar y enlazar las clases para generar el ejecutable .jar, para hacer uso de él se debe tener instalada la herramienta de construcción Maven, puede obtenerse en [22], que es una herramienta de la fundación Apache.

Para la realización del trabajo se ha optado por el uso de la primera opción, es decir, descargar el .zip desde la página oficial y descomprimirlo.

5.2. Instalación del servidor

Una vez obtenido, la instalación es sencilla puesto que todo lo que se necesita es el .jar mencionado anteriormente y la JVM para interpretarlo, pudiéndose ejecutar directamente haciendo uso del script que viene en el .zip dentro de la carpeta bin llamado start.sh, que simplemente llama al interprete java pasándole la clase principal que se debe llamar dentro del .jar de HazelCast.

5.2.1. Primera instalación del servicio

A continuación, se va a detallar el proceso de instalación del servicio. Para este apartado se ha tenido en cuenta que:

1. HazelCast es un servicio escrito completamente en java, por tanto, es un código interpretable, no compilable, por lo que no se puede realizar su compilación. Como tampoco es posible la compilación de un cliente, puesto que ya se ha comentado que no existen y no tiene dependencias, se compilará la JVM, en concreto se ha optado por Openjdk9.

2. HazelCast no está disponible para instalar desde paquetes/repositorios, sin embargo, se han instalado ciertas dependencias necesarias para compilar la JVM mediante este método, que se detallaran en el apartado 5.2.1.2.

5.2.1.1. Instalación desde código fuente (Compilación)

IMPOSIBILIDAD DE COMPILAR EL SERVICIO
Motivo: desarrollado en lenguaje interpretado (java).

Como se ha comentado, el servicio no puede ser compilado y se ha optado por compilar Openjdk9. Se ha utilizado la versión 9 porque es una versión que no se encuentra actualmente en los equipos del laboratorio y así no será necesario desinstalarla previamente.

Para realizar dicha compilación de forma sencilla se ha creado el script "compilar_jvm.sh". Este script se encarga de instalar ciertos programas y dependencias necesarias para llevar a cabo dicha compilación. Una vez ejecutado el script se instalará la nueva versión de JVM en un directorio llamado "jdk9" en el directorio desde el que se ejecute.

Este script instala primero la herramienta "Mercurial" que se encargará de controlar los árboles de versiones para obtener el código fuente (una herramienta de control de versiones), y será necesario para poder clonar el repositorio de código mediante el comando "hg". Después se instalan las dependencias: libXtst-devel libXt-devel libXrender-devel libXi-devel, cups-devel, alsa-lib-devel. Estas dependencias son

necesarias puesto que no se encuentran en los equipos del laboratorio y sin ellas no es posible la compilación del código.

Después, se clona el repositorio, se ejecuta el script "get_source.sh" que obtiene el código necesario del repositorio, se ejecuta el script de configuración "configure", que creará el árbol de directorios necesarios y configurará variables de entorno necesarias, y se compila el código mediante la herramienta make. Por último se verifica que se ha instalado y se realizan algunos test. Detallado:

1. Clonamos los repositorios y obtenemos el código fuente necesario de cada repositorio con el script get_source

   ```
   hg clone http://hg.openjdk.java.net/jdk9/jdk9

   cd jdk9

   bash get_source.sh
   ```

2. Ejecutamos el script de configuracion, creará el arbol de directorios y variables de entorno necesarias.

   ```
   bash configure
   ```

3. Ejecutamos el makefile

   ```
   make images
   ```

4. Se comprueba que se ha instalado correctamente y se realizan algunos test

   ```
   ./build/*/images/jdk/bin/java -version

   make run-test-tier1
   ```

Como anotación de este proceso al ejecutar `make run-test-tier1` en la práctica a veces devuelve error, sin embargo, la JVM está correctamente compilada y de hecho puede utilizarse. Basta con que el comando `./build/*/images/jdk/bin/java -version` devuelva una respuesta correcta. Puede encontrarse más información acerca de este proceso en [23].

Por último, solo falta cambiar desde la version de la JVM que utiliza el equipo actualmente a la versión que se acaba de compilar. Para ello hacemos uso de la heramienta

"alternatives", que se encarga de crear todos los enlaces simbólicos necesarios en la carpeta /etc/alternatives apuntandolos a los ficheros que se han creado en nuestra compilación. Este fue un punto problemático puesto que inicialmente se estaban intentado crear a mano y no se conseguía que funcionase. Se ha creado un script "cambiarjdk.sh" que hace uso de dicha herramienta y nos permite cambiar a la versión que acabamos de compilar. Este script consta simplemente de dos líneas que se muestran a continuación, si se ha ejecutado el script anterior desde un directorio diferente habrá que modificar la primera línea para que corresponda a este:

1. Se añade la carpeta donde se encuentra el código compilado.

```
update-alternatives --install /usr/bin/java java
/home/dit/Trabajo/jdk9/build/linux-x86_64-normal-server-
release/jdk/bin/java 1000
```

2. Se cambia la versión por la que se acaba de añadir.

```
update-alternatives --config java
```

Una vez ejecutado el script se despliega un menú numérico que nos permite seleccionar la versión que queremos usar. En nuestro caso seleccionamos openjdk9 y ya está totalmente instalado y funcional.

5.2.1.2. Instalación desde paquetes/repositorios

Como se ha comentado el servicio no está disponible para su instalación mediante repositorios y tampoco depende de ninguna librería. Sin embargo, se ha hecho uso de la herramienta "yum" para instalar ciertas dependencias desde paquetes/repositorios, necesarias a la hora de compilar la JVM. Los comandos utilizados se muestran a continuación y forman parte del script "compilar_jvm.sh":

Descargar e instalar Mercurial:

```
yum install mercurial
```

Instalamos dependencias para evitar error de cabeceras X11 al ejecutar el script de configuración:

```
yum install libXtst-devel libXt-devel libXrender-devel libXi-devel
```

Instalamos dependencias para evitar error de cups no encontrados:

```
yum install cups-devel
```

Para solucionar el no podemos encontrar alza:

```
yum install alsa-lib-devel
```

Para obtener las códigos de ejemplo se ha necesitado instalar apache maven. Para ello, en el script de configuración del escenario 2 (configurar_executor_service.sh) se ha añadido las siguientes líneas:

Para obtener el repositorio:

```
wget  http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
```

Para instalar apache maven:

```
yum install apache-maven
```

5.2.2. Actualización del servicio

La actualización del servicio es igual de simple que la instalación. Simplemente hay que descargar el .zip de la versión que se desea y descomprimirlo, quedando totalmente funcional ya que se basa en un .jar. En nuestro caso se ha realizado este proceso con la versión 3.10.6, que es la versión justamente anterior a la 3.11 usada, puede encontrarse en [24] haciendo click en “show more”.

Pueden convivir versiones en el sistema siempre que se encuentren en directorios diferentes, ya que, tal como vienen los scripts para iniciar el servidor por defecto, se pueden ejecutar todos los nodos que se deseen en el mismo equipo siempre que se ejecuten desde directorios diferentes, esto es debido a que crean el fichero de lock en el directorio desde el que se ejecuta el script “start.sh”.

5.2.2.1. Actualización desde código fuente

Para actualizar HazelCast desde código fuente, se puede eliminar el actual o dejarlo, clonar desde GitHub la versión que se desee y utilizando de nuevo el comando `mvn clean install` se construirá el .jar de la versión que se haya clonado. Puede encontrarse la versión 3.10.6 en [25], que ha sido la utilizada para este apartado.

Si queremos actualizar o degradar el openjdk9 que habíamos instalado a otra versión, es necesario repetir los pasos de instalación que se han explicado cambiando en el script "compilar_jvm.sh" las líneas:

```
hg clone http://hg.openjdk.java.net/jdk9/jdk9

cd jdk9
```

Por la versión a la que se desee actualizar. En nuestro caso se ha realizado este apartado con la versión jdk8u, por lo que las líneas correspondientes han sido:

```
hg clone http://hg.openjdk.java.net/jdk8u/jdk8u

cd jdk8u
```

Por último se debe modificar en el script "cambiarjdk.sh" la primera para que se incluya en alternatives la versión que se acaba de compilar, en nuestro caso esa línea pasaría a ser:

```
update-alternatives --install /usr/bin/java java
/home/dit/Trabajo/jdk8u/build/linux-x86_64-normal-server-
release/jdk/bin/java 1000
```

Y para finalizar cambiar a la versión de jdk8u cuando aparezca el menu de selección al ejecutar el script "cambiarjdk.sh".

5.2.2.2. Actualización desde paquetes/repositorios

Como HazelCast no está disponible desde repositorios y se ha realizado la instalación de Mercurial mediante la herramienta "yum", se procederá a actualizar esta. Simplemente se debe ejecutar el comando `yum update mercurial`. En caso de que haya alguna actualización disponible se realizará, en otro caso no.

Si queremos actualizar a una versión concreta podemos ejecutar `yum remove mercurial` para eliminarlo, comprobar las versiones disponibles con `yum --showduplicates list mercurial | expand` y por último, instalar la deseada con `yum install mercurial-2.6.2-8.el7_4` donde 2.6.2-8.el7_4 es la versión elegida, y la única que actualmente existe de Mercurial, si existiesen varias podríamos elegir entre las que se listan con el segundo comando mencionado, en este caso al haber solo una la instalación es igual que con `yum install mercurial`.

5.2.3. Desinstalación del servicio

Para desinstalar HazelCast basta con eliminar el directorio donde se encuentra, es decir, el directorio que se genera al descomprimir el .zip descargado, y en nuestro caso, habría que eliminar los scripts de servicios creados de las carpetas /etc/init.d/ y /etc/rc.d/rcX.d/, que se detallarán en el punto 6.

Dado que HazelCast no es compilado y por tanto no se ha instalado desde código fuente ni tampoco está disponible desde repositorios, en los siguientes apartados pasaremos a describir como se desinstala Openjdk9, Mercurial y Maven, ya que estos son los componentes que hemos instalado desde código fuente y desde repositorios respectivamente.

5.2.3.1. Desinstalación desde código fuente

Para la desinstalación de Openjdk9 debemos volver a utilizar el script "cambiarjdk.sh" y seleccionar otro JRE disponible en el sistema. Después, basta con eliminar el directorio jdk9 que había creado el script "compilar_jvm.sh". El proceso debe ser en este orden, ya que si eliminamos el directorio y los enlaces simbólicos de /etc/alternatives siguen apuntado a él, se producirá un error si se utiliza alguna aplicación java, hasta que se cambien estos enlaces.

5.2.3.2. Desinstalación desde paquetes/repositorios

Para la desinstalación de Mercurial desde repositorios basta ejecutar el comando `yum remove mercurial` y este quedará desinstalado del sistema. E igual con Maven `yum remove Maven`.

5.3. Configuración del servidor

HazelCast puede ser configurado mediante ficheros de configuración (declarativamente) o mediante código a través de la API de configuración (de forma programada) o de una forma mixta. Para la realización de este trabajo nos centraremos en la configuración mediante ficheros de configuración, es decir, declarativamente.

A continuación, se muestran los ficheros de configuración de HazelCast y se describe el uso de cada uno de ellos:

- **hazelcast.xml**: es el fichero de configuración principal, en él se configuran todos los parámetros de comportamiento del servidor. La ruta de este fichero viene

determinada por el parámetro `-Dhazelcast.config=/ruta/myhazelcast.xml` que debe pasarse en la llamada al ejecutable .jar. Sin embargo, si no se le pasa utiliza un fichero de configuración por defecto. Dependiendo del uso que se le dé a HazelCast el fichero podrá situarse en una ubicación u otra. Por ejemplo, para la integración con Tomcat este fichero debe situarse en `$CATALINA_HOME/lib/` ya que ahí será buscado por el ejecutable según la referencia. Por ejemplo, un parámetro de configuración sería:

```
<multicast enabled="true">

        <multicast-group>224.2.2.3</multicast-group>

        <multicast-port>54327</multicast-port>

</multicast>
```

Que sirve para configurar la ip y puerto del autodescubrimiento de nodos por grupo multicast. En el Anexo A se detallará

el contenido configurado en este fichero, sobre todo todos los parámetros que han sido utilizados. Puede obtenerse más información acerca del fichero en [26].

- **Cluster.sh**: este script una vez iniciado el servidor nos permite administrar el clúster. Permite obtener el estado del clúster, cambiar el estado, obtener la versión, cambiar la versión, apagarlo y forzar el limpiado de datos y rearrancar. Se puede obtener toda la información acerca de este script y sus parámetros en [27].

- **Healthcheck.sh**: para usar este script hay que establecer en hazelcast.xml la propiedad `hazelcast.http.healthcheck.enabled` a "true". Cuando se ejecuta este script hace uso de la API REST que ofrece el servidor HazelCast de forma que el contenido que devuelve es el mismo que si accedemos desde un navegador a `http://IP:Puerto/hazelcast/health`. Este script nos ofrece información sobre el clúster o el nodo solicitado, como el estado del nodo o del clúster, el tamaño del clúster, etc. Puede obtenerse toda la información sobre este script en [28].

6. Puesta en funcionamiento del servicio

Tras descomprimir el archivo .zip, dentro de la carpeta hazelcast-3.11/bin/ existe un script llamado "start.sh". Si ejecutamos este script desde un terminal buscará la aplicación java en el equipo y ejecutará el servidor con la configuración existente en el archivo hazelcast.xml de la misma carpeta. Además, creará un fichero de bloqueo, también en la misma carpeta, que permitirá que no se inicien varias instancias desde el mismo directorio, sino que para iniciar varias instancias es necesario descomprimir el .zip en directorios diferentes.

Así mismo, existe un script llamado "stop.sh", que mata el proceso y elimina el fichero de bloqueo.

Con estos dos scripts es suficiente para arrancar el servidor de forma simple y están disponibles tan solo descomprimiendo el .zip descargado. Si se arrancan más de un servidor se auto-descubrirán de la forma en que esté configurado en hazelcast.xml y formarán un clúster, así se irán uniendo nodos al clúster conforme se inicien más instancias del servicio, en la misma o en máquinas diferentes.

Pero en nuestro caso, se ha realizado la modificación de los mismos para arrancar el servicio con el sistema.

6.1. Arranque del servicio con el sistema

Para el arranque del servicio con el sistema se ha creado el script de servicio llamado "hazelcast" y se ha realizado mediante SysVinit.

Ha sido necesario modificar el script "start.sh" que trae el .zip por defecto, en concreto, se ha eliminado la creación del fichero de bloqueo en la misma carpeta y se ha almacenado el PID del proceso que se crea en /var/run/hazelcast.pid para poder matarlo posteriormente con "killproc". Además, en la llamada al ejecutable se ha redireccionado tanto la salida de errores como la estándar hacia el fichero /var/log/hazelcast para poder visualizar la salida del Health Monitor, que se imprime en salida estándar y si no hacemos esto no podemos verlo. El Health Monitor se explicará en el apartado 6.2.2. También, se ha modificado la variable "HAZELCAST_HOME" para apuntarla directamente al directorio donde se encuentra el .jar de HazelCast, ya que este estaba siendo buscado a partir de la ruta donde inicialmente se encontraba este script y el script ha sido cambiado de sitio al directorio

/home/dit/Trabajo. Además, se ha realizado la llamada al ejecutable del servidor con la opción:

```
-Dhazelcast.config=/home/dit/Trabajo/hazelcast.xml
```

Que configura la ruta donde encontrar el archivo de configuración "hazelcast.xml", y la opción:

```
-Dhazelcast.diagnostics.directory=/home/dit/Trabajo/logs
```

Que configura la ruta para los ficheros de logs, también se ha utilizado otras opciones para la configuración de ficheros de logs que se muestran en el apartado 6.2.1.

En el script de servicio "hazelcast" se han implementado las funciones "start", "stop", "status", "reload" y "restart".

Para la función "start" se ha comprobado la existencia del fichero de bloqueo en /var/lock/subsys/ y en caso de no existir se ha llamado al script "start.sh" que hemos modificado para iniciar el proceso. Una vez iniciado se ha creado el fichero de bloqueo en /var/lock/subsys/.

La función "stop" hace uso de la función "killproc" para matar al proceso mediante el PID que habíamos almacenado en el fichero /var/run/hazelcast.pid y elimina el fichero de bloqueo /var/lock/subsys/hazelcast.

La función "status" hace uso de la función "status" para devolver el estado del servidor.

Las funciones "reload" y "restart" hacen uso de las funciones anteriores "stop" y "start" para re-arrancar el servicio, en ese orden.

Además, se ha creado el script "instalarHazelcast.sh" que sitúa el script de servicio "hazelcast" en /etc/rc.d/init.d/ y hace uso del comando "chkconfig" para crear los enlaces simbólicos en /etc/rd.d/rcX.d/ y configurar los niveles de ejecución donde el servicio se iniciará o se parará. Por último, este script da los permisos necesarios al script de servicio.

Por lo tanto, para instalar el servicio y que se arranque con el sistema basta con ejecutar:

```
sh instalarHazelcast.sh #instala el servicio con SysVinit

service hazelcast restart #para arrancar el servicio
```

6.2. Administración y monitorización del funcionamiento

HazelCast ofrece varias posibilidades para monitorizar y comprobar el correcto funcionamiento del servicio. Las principales son: ficheros de logs y health monitor. Para los logs ofrece una gran cantidad de "plugins" que básicamente son componentes de log, y que pueden encontrarse en [29]. En los siguientes subapartados se describirán cada una de ellas.

6.2.1. Configuración y Uso de los ficheros de registro

Para configurar la creación de ficheros de registros es necesario llamar al ejecutable pasándole el parámetro:

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

En nuestro caso se ha usado:

```
-Dhazelcast.diagnostics.directory=/home/dit/Trabajo/logs
```

Por defecto se crean un máximo de 10 ficheros de 50MB, pero puede configurarse el número de ficheros y el tamaño mediante los parámetros:

```
-Dhazelcast.diagnostics.max.rolled.file.size.mb=100

-Dhazelcast.diagnostics.max.rolled.file.count=5
```

Eso configurará el número, tamaño y directorio de los archivos de log, pero todavía es necesario activar la creación de los logs, lo que se hace con los parámetros :

```
-Dhazelcast.diagnostics.enabled=true
-Dhazelcast.diagnostics.metric.level=info
-Dhazelcast.diagnostics.invocation.sample.period.seconds=30
-Dhazelcast.diagnostics.pending.invocations.period.seconds=30
-Dhazelcast.diagnostics.slowoperations.period.seconds=30
-Dhazelcast.diagnostics.storeLatency.period.seconds=60
```

Como se ha comentado todos estos parámetros son necesarios en la llamada al ejecutable del servidor, es decir, en nuestro caso en el script "start.sh" que habíamos modificado.
Las opciones de registro que ofrece son las que aparecen en los llamados "plugins" que pueden verse en [29]. Algunos de estos son: BuidInfo, HazelCast Instance, Invocation, etc.

El contenido del fichero de log creado contendrá varias líneas del tipo:

```
<Date> PluginName[
        <log content for diagnostics plugin>]
```

Donde date es la fecha, PluginName es el nombre del plugin que realiza el log, y entre los corchetes irá la información de log del plugin correspondiente.

6.2.2. Arranque del servicio en modo detallado (verbose)

Lo más parecido a un modo detallado que ofrece HazelCast es el Health Monitor. El Health Monitor imprime periódicamente en la consola información sobre el estado de los nodos.

El Health Monitor se configura mediante propiedades del sistema, las propiedades del sistema en HazelCast se establecen en el fichero de configuración "hazelcast.xml" dentro de la etiqueta `<properties>` y por cada propiedad se debe poner una etiqueta `<property>`.

Las propiedades de configuración correspondientes al HealthMonitor son:

- `hazelcast.health.monitoring.delay.seconds` establece cada cuantos segundos se imprime la información sobre los nodos. Por defecto, 30 segundos.
- `hazelcast.health.monitoring.level` establece el nivel de detalle de la información que se registra. Puede ser OFF (se desactiva), NOISY (se imprime toda la información posible), SILENT (es el valor por defecto e imprime información cuando se sobrepasan ciertos umbrales de nivel de CPU y memoria consumida). Por defecto, SILENT.
- `hazelcast.health.monitoring.threshold.memory.percentage` establece el umbral de porcentaje de memoria consumido para el nivel NOISY. Por defecto, 70.
- `hazelcast.health.monitoring.threshold.cpu.percentage` establece el umbral de porcentaje de CPU consumido para el nivel NOISY. Por defecto, 70.

Por ello, lo que correspondería con el modo "verbose" sería configurarlo en nivel NOISY, en nuestro caso cada 2 segundos, con ambos umbrales al 50%. Se muestra un fragmento del fichero de configuración usado durante las pruebas del servicio:

```
<properties>

      <!—otras propiedades -->

      <!-- Configuacion del health monitor-->
```

```
      <property
name="hazelcast.health.monitoring.delay.seconds">2</property>

      <property name="hazelcast.health.monitoring.level">NOISY</property>

      <property
name="hazelcast.health.monitoring.threshold.memory.percentage">50</proper
ty>

      <property
name="hazelcast.health.monitoring.threshold.cpu.percentage">50</property>

    </properties>
```

La información que imprime el Health Monitor puede verse en /var/log/hazelcast puesto que se ha redireccionado la salida del comando que llama al ejecutable del servidor hacia ese fichero.

6.2.3. Seguridad del servicio

HazelCast ofrece varias opciones para garantizar la seguridad tanto de los miembros del clúster como de los clientes, aunque solo una en la versión libre.

- Comencemos por la seguridad para gestionar el clúster mediante el portal de gestión. Para ello es necesario configurar las siguientes líneas en "hazelcast.xml" correspondientes a usuario y contraseña del "grupo", que sería el clúster:

  ```
  <group>

        <name>dev</name>

         <password>dev-pass</password>

  </group>
  ```

  Aunque según la página oficial el campo password será eliminado en futuras versiones, por lo que la seguridad que esto proporciona es bastante débil, basta con saber el nombre del grupo para romperla, es algo parecido a lo que ocurre con el protocolo SNMP.

  Esta es la única medida de seguridad que podemos configurar puesto que el resto requieren Hazelcast Enterprise Edition que es de pago.

- Una de las opciones de pago es JAAS, con la que podemos mejorar la seguridad entre miembros del clúster, y miembros del clúster y clientes. Además, permite realizar verificaciones de control de acceso en las operaciones de los clientes. Aunque esta opción **no podemos utilizarla** en el desarrollo del trabajo porque requiere utilizar HazelCast Enterprise Edition que es de pago. Para ello hay que configurar en "hazelcast.xml" las siguientes líneas:

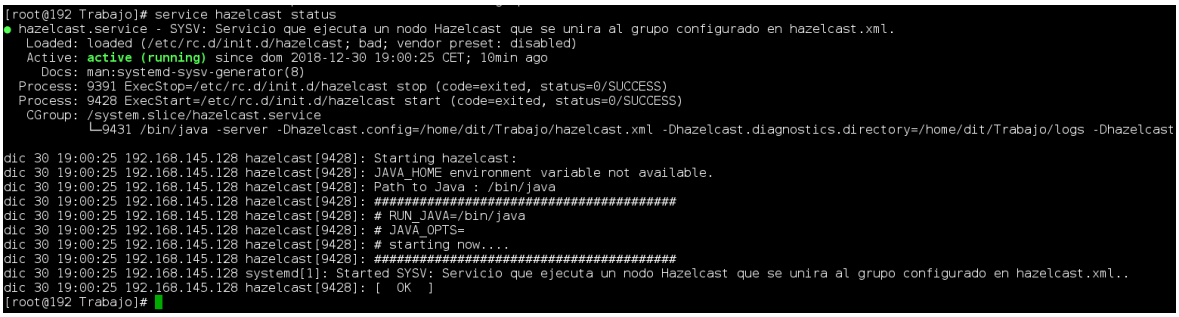
```
<security enabled="true">
   ...
</security>
```

  En los puntos suspensivos se colocan las opciones deseadas.

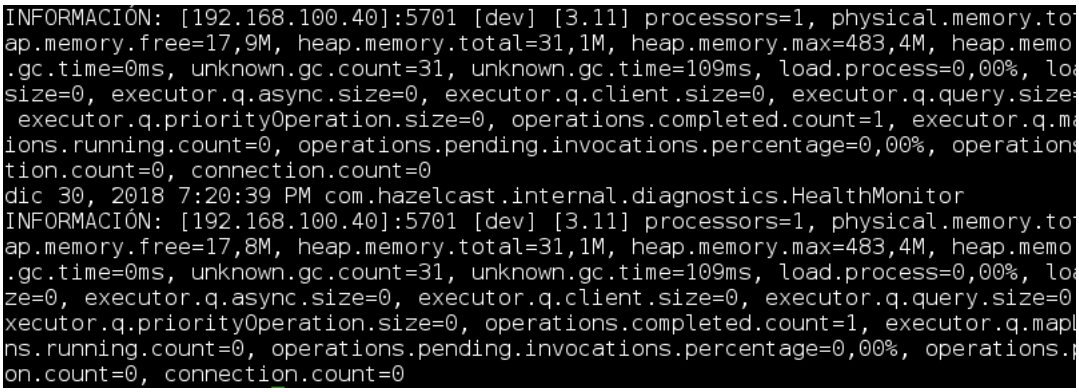
- El resto de opciones de seguridad, como por ejemplo el uso de ssl, se pueden encontrar en [30]. No se profundizará en su explicación puesto que no es posible probarlas ya que se necesita la versión de pago, y no se ha encontrado ninguna forma de utilizarlas mediante otras herramientas.

6.3. Tests y Pruebas del correcto arranque del servicio

Una vez instalado el servicio como servicio SysVinit y configurado para arrancarse con el sistema en el nivel de ejecución 5, que es como se ha configurado, podemos comprobar el correcto arranque del mismo ejecutando `service hazelcast status`, que nos devolverá si se ha arrancado correctamente, comprobación:

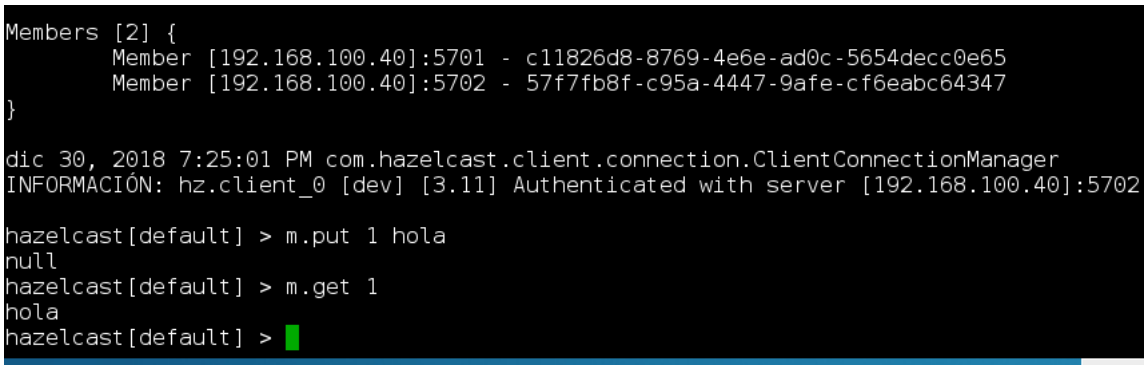


También podemos comprobarlo viendo el fichero de logs creados mediante `cat /var/log/hazelcast`, se muestra un fragmento ya que la salida es bastante amplia:



La imagen anterior muestra información que el Health Monitor ha impreso en la salida estandar, es decir, que el servicio se ha iniciado correctamente y el Health Monitor está funcionando correctamente.

La forma más fácil de comprobarlo es utilizar el cliente de prueba ejecutando el script "clientConsole.sh" que lanza un cliente que se conecta con el clúster. Si la conexión no falla y podemos introducir y obtener datos quiere decir que el servicio funciona correctamente, además nos mostrará el número de quipos que forman el clúster. A continuación, se muestra una captura de introducción y obtención de datos:



En la imagen anterior se puede ver como existen dos nodos unidos al clúster y se introduce el valor "hola" con la clave "1", posteriormente se obtiene el objeto con clave "1" que devuelve el valor "hola" que era el introducido previamente.

También podemos hacer uso del script "healthcheck.sh" que proporciona HazelCast para ver el estado del clúster. Se muestra la salida a continuación:

```
[dit@192 bin]$ sh healthcheck.sh
Hazelcast::NodeState=ACTIVE
Hazelcast::ClusterState=ACTIVE
Hazelcast::ClusterSafe=TRUE
Hazelcast::MigrationQueueSize=0
Hazelcast::ClusterSize=2
[dit@192 bin]$
```

Se puede ver que el tamaño del clúster son 2 nodos, que está activo, etc. Todo esto quiere decir que funciona correctamente.

HazelCast al arrancar abre un único proceso, correspondiente a la ejecución del archivo .jar. Además, puede pertenecer a cualquier usuario, habrá que tener en cuenta en que puerto se desea poner a escuchar el servidor. Si es menor del 1024 se necesitará ser root para usarlo, en otro caso no. Si que puede ser que cree varios hilos para realizar tareas de ejecución distribuida, en el segundo escenario se aclarará esto. El puerto se configura con la directiva <port> en "hazelcast.xml" por ejemplo: `<port auto-increment="true" port-count="100">5701</port>`. En nuestro caso al necesitar hacer uso del comando `service` el sistema nos solicita ser root para iniciarlo, pero si se inicia directamente desde el ejecutable, o con los scripts que vienen por defecto no será necesario ser root. Se muestra a continuación la salida del comando `ps uxa | grep hazelcast` que nos mostrará procesos y usuarios asociados pertenecientes a HazelCast:

```
[root@192 Trabajo]# ps uxa | grep hazelcast
root      9431  2.6  6.9 2589200 141436 ?      Sl   19:00   1:04 /bin/java -server -Dhazelcast.config=/home/dit/Trabajo/hazel
 -Dhazelcast.diagnostics.enabled=true -Dhazelcast.diagnostics.metric.level=info -Dhazelcast.diagnostics.invocation.sample.per
.seconds=30 -Dhazelcast.diagnostics.slowoperations.period.seconds=30 -Dhazelcast.diagnostics.storeLatency.period.seconds=60 c
dit       9644  2.5  5.9 2592944 121544 pts/1  Sl   19:24   0:25 /usr/bin/java -server com.hazelcast.core.server.StartServer
```

Puede verse que tenemos dos procesos asociados a hazelcast, esto se debe a que tenemos dos instancias del servidor corriendo en la mima máquina. Una, la asociada al usuario root, es la que se inicia con el sistema. La otra, asociada al usuario dit, ha sido ejecutada mediante los scripts que ofrece HazelCast por defecto. Se ha hecho así para demostrar que cualquier usuario puede ejecutar una instancia del servidor. La única limitación es el puerto, y en nuestro caso, que debemos ser usuario root para usar `service`. No existe un parámetro de configuración asociado a configurar el usuario como existía la directiva "User" en Apache. Simplemente el proceso lo abrirá el usuario que llame al ejecutable.

En cuanto a los puertos, ahora tenemos una sola instancia de HazelCast configurada para que use el puerto tcp 5701 para comunicación entre nodos y con clientes, y el puerto udp 54327 para descubrimiento de nuevos nodos con ip multicast. Puede verse que está así configurado en la siguiente imagen:

```
<network>
    <port auto-increment="true" port-count="100">5701</port>
    <outbound-ports>
        <!--
        Allowed port range when connecting to other nodes.
        0 or * means use system provided port.
        -->
        <ports>0</ports>
    </outbound-ports>
    <join>
        <multicast enabled="true">
            <multicast-group>224.2.2.3</multicast-group>
            <multicast-port>54327</multicast-port>
        </multicast>
```

Para comprobar que es correcto y que funciona así se ha usado los comandos `netstat -at` para mostrar los puertos tcp y `netstat -au` para mostrar los puertos udp. La salida de estos comandos que confirman que los puertos son los comentados de muestra a continuación:

```
[dit@192 Trabajo]$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:5701            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 192.168.145.128:5702    192.168.145.128:38249   TIME_WAIT
[dit@192 Trabajo]$ netstat -au
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 192.168.145.128:ntp     0.0.0.0:*
udp        0      0 192.168.145.128:ntp     0.0.0.0:*
udp        0      0 localhost:ntp           0.0.0.0:*
udp        0      0 0.0.0.0:ntp             0.0.0.0:*
udp        0      0 0.0.0.0:53653           0.0.0.0:*
udp        0      0 localhost:41500         localhost:ntp           ESTABLISHED
udp        0      0 0.0.0.0:50930           0.0.0.0:*
udp        0      0 0.0.0.0:54327           0.0.0.0:*
udp        0      0 0.0.0.0:bootpc          0.0.0.0:*
udp        0      0 0.0.0.0:bootpc          0.0.0.0:*
udp6       0      0 [::]:ntp                [::]:*
udp6       0      0 [::]:6683               [::]:*
udp6       0      0 [::]:15127              [::]:*
```

Explicando un poco más la `línea <port auto-increment="true" port-count="100">5701</port>` la propiedad “auto-increment” permite que si se utilizan varias instancias del servidor en el mismo equipo y el puerto 5701 configurado ya está en uso se vaya incrementando de uno en uno hasta encontrar uno libre. La propiedad “count” establece el número máximo de puertos que pueden existir para instancias hazelcast. En resumen, tal como lo hemos configurado el servidor en su inicio buscará un puerto libre entre 5701 y 5801. Remarcar que los puertos 5701-5801 son de escucha, para enviar datos a otro nodo, un nodo HazelCast utiliza un puerto normal como si fuera cliente, en caso de que sea el que abre la comunicación, contra el puerto 5701 del otro servidor (o el que corresponda). En la respuesta del otro servidor si aparecerá puerto origen 5701 (o el que

corresponda). Los puertos usados para conectar con otros nodos se configuran con las siguientes directivas:

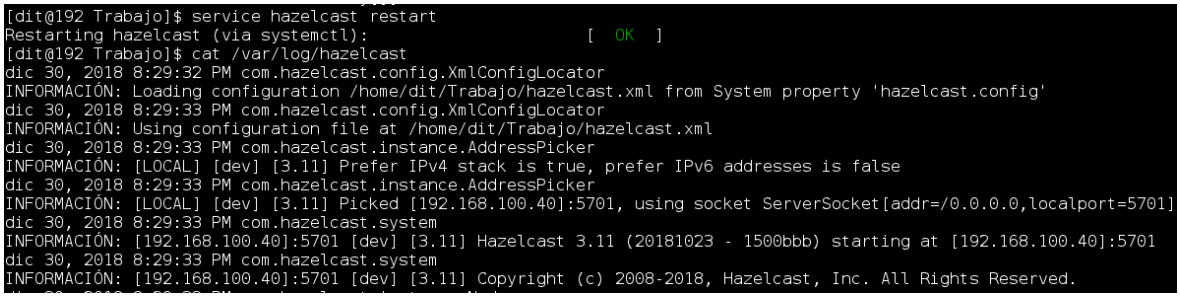
```
<outbound-ports>

            <ports>0</ports>

</outbound-ports>
```

El valor 0 o * significa que usará el asignado por el sistema.

Para comprobar que los logs se generan correctamente al arrancar no tenemos más que `ejecutar cat /var/log/hazelcast` y nos aparecerá la información relativa al arranque ya que estamos usando el modo NOISY. Se muestra parcialmente esta información ya que es muy amplia:



Como vemos en la imagen anterior, nos indica la ruta de la que se toma el archivo de configuración “hazelcast.xml”, que se hace uso de direcciones ipv4, la dirección y puerto donde escucha el servidor, etc.

7. Diseño de los escenarios de prueba

A continuación, se presentará los escenarios diseñados para la defensa del trabajo. Para este apartado tener en cuenta que el escenario más complejo es el primero descrito, mientras que el segundo es más simple. Esto se ha hecho así porque en el momento de redacción de este apartado no se tenía claro cual sería el segundo escenario.

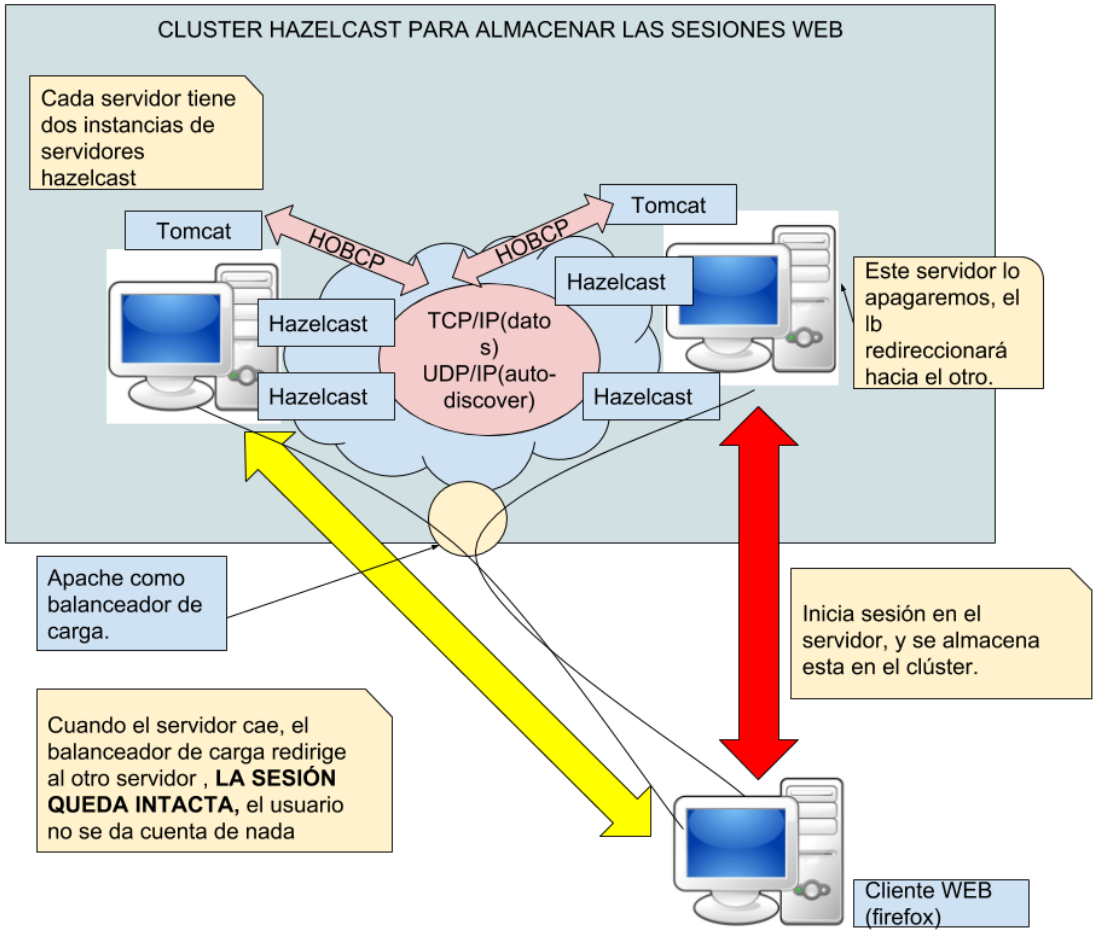
7.1. Escenario de Defensa 1: Web Session Clustering, integración con Tomcat y uso de Apache como balanceador de carga.

Tal como se indica en el título, en este escenario se va a utilizar la característica de HazelCast de Web Session Clustering, para lo que se hará uso de la integración con Tomcat y se utilizará Apache como balanceador de carga.

Para la realización de este escenario se ha descargado de [19] una pequeña aplicación web en formato .war que permite realizar el log in y el log out de un usuario, almacenando la sesión de este durante 30 segundos.

La sesión del usuario quedará almacenada en el clúster, formado por ambos equipos del laboratorio con el servidor HazelCast, por lo que cuando paremos el servidor Tomcat al que está conectado el navegador web, automáticamente se recuperará la sesión web en el servidor Tomcat del otro equipo, que la obtendrá del clúster. Para realizar estas pruebas, ya que con cada servidor Tomcat se lanzará otra instancia del servidor HazelCast para conectarse como clientes a dicha instancia, se tendrán un total de 4 miembros en el clúster de HazelCast, los 2 arrancados con el sistema gracias a los scripts de servicio y los 2 que se arrancarán con Tomcat, uno por cada servidor Tomcat.

7.1.1. Escenario 1: Esquema de la red



Tener en cuenta que el cliente web Firefox se encontrará también en uno de los dos equipos superiores puesto que solo se dispone de dos equipos en el laboratorio, se ha separado por simplicidad del dibujo. Lo mismo ocurre con el balanceador de carga Apache, estará en uno de los equipos, pero se dibuja fuera por simplicidad.

Para configurar el escenario se ha creado un script llamado "configurar_web_session_cluster_tomcat.sh".

7.1.2. Escenario 1: Configuración del servidor

La configuración realizada es la siguiente:

En el fichero “hazelcast.xml” se han configurado los puertos como se comentó en el apartado 6.3. Se ha configurado el autodescubrimiento multicast con las siguientes líneas:

```
      <multicast enabled="true">

            <multicast-group>224.2.2.3</multicast-group>

            <multicast-port>54327</multicast-port>

       </multicast>
```

Se ha configurado el Health monitor y el Heathcheck con las siguientes líneas:

```
<properties>

        <property
name="hazelcast.http.healthcheck.enabled">true</property>

      <property name="hazelcast.rest.enabled">true</property>

      <property
name="hazelcast.client.statistics.enabled">true</property>

      <!-- Configuacion del health monitor-->

      <property
name="hazelcast.health.monitoring.delay.seconds">2</property>

      <property name="hazelcast.health.monitoring.level">NOISY</property>

      <property
name="hazelcast.health.monitoring.threshold.memory.percentage">50</proper
ty>

      <property
name="hazelcast.health.monitoring.threshold.cpu.percentage">50</property>

    </properties>
```

Se ha configurado también el centro de mantenimiento y el grupo, la ip del centro de mantenimiento depende del equipo donde se ejecute este (es importante que todos los nodos de un clúster tengan configurado el mismo centro de mantenimiento, sino se producen errores), con las siguientes líneas:

```
<group>

        <name>dev</name>
```

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

    </group>

    <management-center enabled="true" update-
interval="3">http://172.16.17.12:8080/mancenter/</management-center>
```

También, en la llamada al ejecutable se han activado las opciones para la creación de ficheros de logs que se comentaron el apartado 6.2.1.

Además, es necesario copiar el ejecutable de HazelCast (hazelcast-all-3.11.jar), el gestor de sesiones web para Tomcat (hazelcast-tomcat7-sessionmanager-1.1.3.jar) y el archivo de configuración (hazelcast.xml) en $CATALINA_HOME/lib/ para lo que el script "configurar_web_session_cluster_tomcat.sh" ejecuta los siguientes comandos, en ambos equipos:

```
cp ./hazelcast-3.11/lib/hazelcast-all-3.11.jar /usr/share/tomcat/lib/

cp ./hazelcast-tomcat7-sessionmanager-1.1.3.jar /usr/share/tomcat/lib/

cp ./hazelcast-3.11/bin/hazelcast.xml /usr/share/tomcat/lib/
```

También es necesario modificar el archivo "context.xml" de Tomcat, añadiendo una línea con un manager para HazelCast:

```
<Manager className="com.hazelcast.session.HazelcastSessionManager" />
```

Además, necesitamos modificar el archivo "mod_jk_workers.properties" de apache para crear el balanceador de carga y los workers que enviarán las peticiones a los servidores Tomcat, solo en uno de los equipos, añadiendo las siguientes líneas:

```
worker.list=tomcat, tomcat2, balancer

#creando balanceador de carga y dos workers para los dos equipos.

worker.balancer.type=lb

worker.balancer.balance_workers=tomcat,tomcat2


worker.tomcat.port=8009

worker.tomcat.host=172.16.17.12

worker.tomcat.type=ajp13
```

```
worker.tomcat2.port=8009

worker.tomcat2.host=172.16.17.11

worker.tomcat2.type=ajp13
```

Se debe modificar también el archivo "mod_jk.conf" de apache en el equipo donde vaya a configurarse el balanceador de carga añadiendo las siguientes líneas, para redireccionar las peticiones que vayan a nuestra aplicación web al balanceador de carga, que la enviará a uno de los servidores Tomcat:

```
JkMount /httpSession/* balancer
```

Y en el mismo fichero, como hemos pondremos el .war de la aplicación en el directorio base webapps:

```
JkAutoAlias /usr/share/tomcat/webapps

   <Directory /usr/share/tomcat/webapps>

        Order allow,deny

        Allow from all

   </Directory>
```

En el fichero de configuración de Tomcat "server.xml" es necesario añadir un listener para hazelcast mediante la línea:

```
<Listener className="com.hazelcast.session.P2PLifecycleListener" />
```

Establecer el mecanismo y puerto por el que se recibirán las solicitudes que vengan de Apache:

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

Y configurar un virtualhost con el directorio base, que será el virtual host por defecto:

```
<Host      name="localhost"      appBase="webapps"      unpackWARs="true"
autoDeploy="true">
```

Así mismo hacemos uso de un fichero "httpd.conf" de configuración en el que tenemos una configuración básica. Y además tenemos que copiar el fichero "mod_jk.so" en /etc/httpd/modules/:

```
cp ./mod_jk.so /etc/httpd/modules/mod_jk.so
```

Es necesario modificar los permisos de los archivos comentados para que Tomcat pueda utilizarlos, mediante los comandos:

```
chmod 755 /usr/share/tomcat/lib/hazelcast-tomcat7-sessionmanager-1.1.3.jar

chmod 755 /usr/share/tomcat/lib/hazelcast-all-3.11.jar

chmod 644 /usr/share/tomcat/lib/hazelcast.xml
```

Y, por último, situar nuestra aplicación web que hace uso de sesiones en el directorio $CATALINA_HOME/webapps/ y darle permisos necesarios mediante los comandos:

```
cp ./httpSession.war /usr/share/tomcat/webapps/

chmod 744 /usr/share/tomcat/webapps/httpSession.war
```

Tras realizar todo esto, y reiniciar los servidores, que puede verse de forma completa en el script "configurar_web_session_cluster_tomcat.sh" que se ha creado, las sesiones web que cree nuestra aplicación web serán alamacenadas en el clúster HazelCast de forma que mientras un nodo del clúster siga activo y uno de los servidores Tomcat siga activo, podrán recuperarse todas las sesiones web sin pérdida cuando se produzcan caídas de nodos.

Nota: las direcciones ip que aparecen en los ficheros de configuración dependen de las de los equipos que se estén utilizando.

7.1.3. Escenario 1: Tests y Pruebas del Escenario

Para comprobar que funciona correctamente, las operaciones a realizar son las siguientes:

1. Iniciar los servidores y un navegador web.
2. Acceder a la ip y puerto de Apache (http://172.16.17.12:80/httpSession en nuestro caso) con Firefox para acceder a la página de inicio de sesión.
3. Iniciar sesión con usuario "candidjava" y password vacío. Se creará una sesión que se almacenará en el clúster HazelCast.

4. Apagamos el servidor Tomcat con `service tomcat stop` comprobamos que la sesión sigue activa, pues el otro servidor Tomcat la obtiene del clúster si no era él el que la tenía.

5. Arrancamos de nuevo el servidor Tomcat con `service tomcat start` y apagamos el del otro equipo, para asegurar que antes no se mantuvo porque estaba en el servidor al que estábamos conectados. Comprobamos que la sesión no se ha perdido en ningún momento.

En resumen, la idea de esta comprobación es asegurarnos de que cuando se mantiene la sesión no sea porque hemos apagado el servidor al que no estábamos conectados, por ello lo de apagarlos alternativamente.

Cuando se apague el servidor Tomcat, uno de los 4 miembros HazelCast del clúster caerá, repartiéndose de nuevo las particiones, y lo contrario ocurre cuando se vuelve a encender, que se añade un nuevo miembro y se reparten de nuevo las particiones.

Otra forma más simple de asegurarnos que funciona es:

1. Arrancar un único servidor Tomcat para asegurarnos de que nos conectamos a ese a través del balanceador de carga.

2. Conectarnos mediante un navegador web (a la ip y puerto de apache) e iniciar sesión. El objeto de sesión se guardará en el clúster.

3. Arrancar el otro servidor Tomcat, se agregará un nuevo miembro al clúster y se repartirán particiones.

4. Paramos el servidor Tomcat inicial al que estábamos conectados con `service tomcat stop`.

5. Recargamos la página web comprobando que la sesión sigue activa. En caso de que falle y la sesión no esté activa nos enviará de nuevo a la página de inicio de sesión.

7.1.4. Escenario 1: Análisis del intercambio real de mensajes de red

Se presentan los mensajes intercambiados por los protocolos que utiliza HazelCast. En este caso, se están realizando las pruebas con dos máquinas virtuales con direcciones 192.168.100.40 y 192.168.100.97.

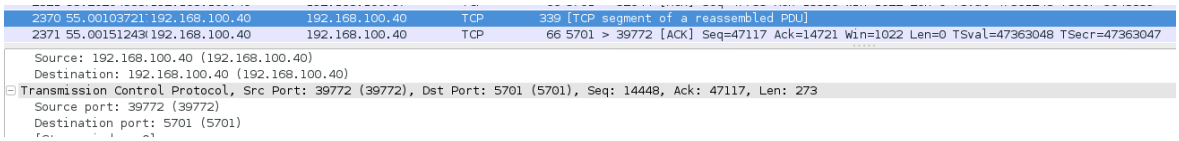
El autodescubrimiento lo hemos configurado con dirección ip multicast 224.2.2.3 en el puerto 54327. Se muestra la secuencia de mensajes de autodescubrimiento capturada:

6190	9.730358083	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
6191	9.730372288	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
6192	9.730618439	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
6193	9.730837223	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
6194	9.731059303	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
6195	9.731266602	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
6196	9.870343534	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
6197	9.870361305	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
6198	9.870618190	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
6199	9.870869380	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
6200	9.870968701	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
6201	9.871399676	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
10131	19.59593750(	192.168.100.40	224.2.2.3	UDP	353	Source port: 54327 Destination port: 54327
10132	19.59774388:	192.168.100.40	224.2.2.3	UDP	353	Source port: 54327 Destination port: 54327

En cuanto a la comunicación entre miembros del clúster para distribuir información, cuando se realiza el inicio de sesión y se almacenan nuevos datos en el clúster (en este caso el objeto de sesión) podemos observar una secuencia de mensajes TCP como la siguiente entre los miembros del clúster (la secuencia es mucho más larga se muestra un extracto):

182	12.51950523(	192.168.100.97	192.168.100.40	TCP	339 [TCP segment of a reassembled PDU]
183	12.51953507!	192.168.100.40	192.168.100.97	TCP	66 56183 > 5701 [ACK] Seq=18381 Ack=8792 Win=1016 Len=0 TSval=45231027 TSecr=6913639
184	12.53241680(	192.168.100.97	192.168.100.40	TCP	1482 [TCP segment of a reassembled PDU]
185	12.53244987(	192.168.100.40	192.168.100.97	TCP	66 56183 > 5701 [ACK] Seq=18381 Ack=10208 Win=1016 Len=0 TSval=45231040 TSecr=6913651

Ahora, mostraremos un mensaje TCP que sería del protocolo HOBCP, protocolo que wireshark no reconoce. Se sabe que es un mensaje que usa este protocolo porque podemos ver en los datos del mismo que tiene misma dirección ip origen y destino, lo que significa que es un mensaje que va desde Tomcat al clúster HazelCast con el objeto de sesión y además tiene puerto destino 5701 (puerto de escucha configurado en hazelcast) y puerto origen distinto (puerto que utiliza el cliente):

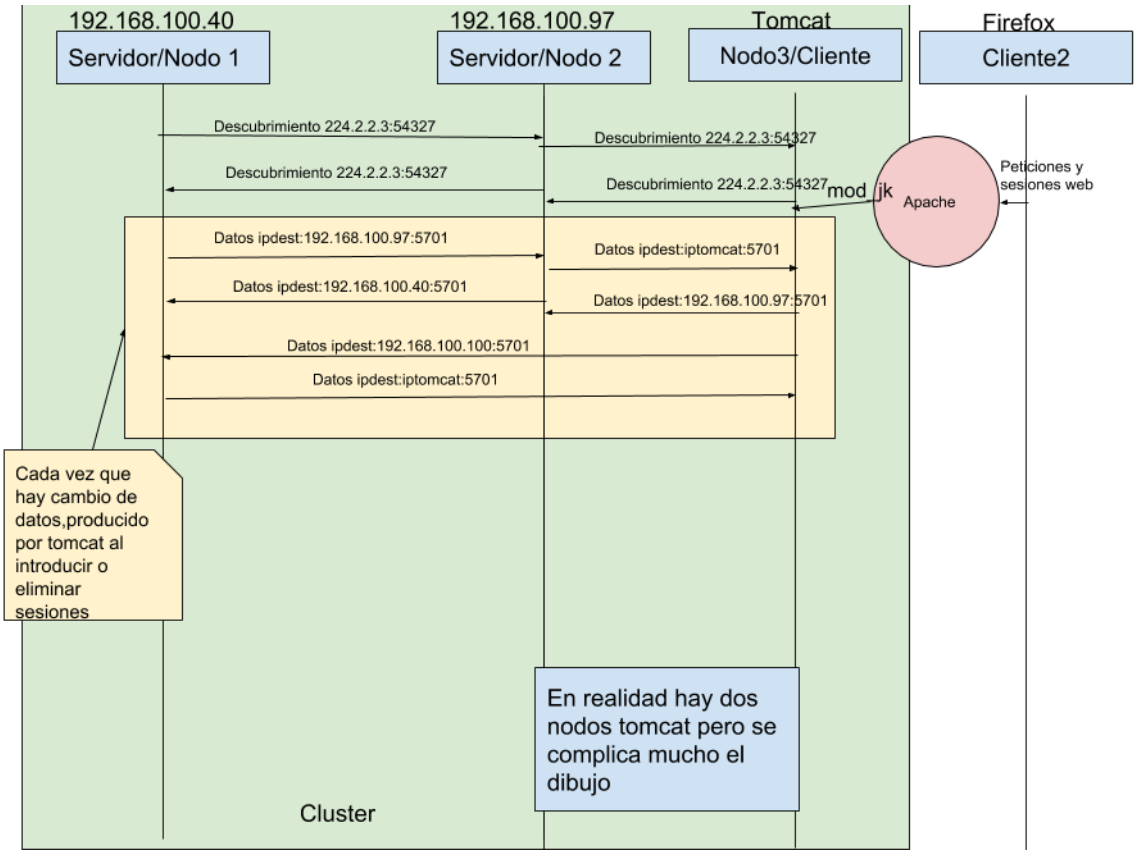


Igual que antes la secuencia de mensajes es más larga se muestra solo un y su ack y los datos correspondientes a los puertos. Muestro ahora un mensaje del mismo tipo, pero en vez de entre cliente y servidor del mismo equipo, estando estos en equipos diferentes:

2284	51.33098349	192.168.100.97	192.168.100.40	TCP	66 55800 > 5701 [ACK] Seq=18943 Ack=50387 Win=1021 Len=0 TSval=9041989 TSecr=47359377
2285	51.36355897(	192.168.100.97	192.168.100.40	TCP	339 [TCP segment of a reassembled PDU]

Puede verse las direcciones ip de equipos distintos, puerto origen 55800 que sería el que usa el cliente asignado por el sistema y puerto destino el de HazelCast 5701.

Aclarar que, entre servidores HazelCast el que inicia la comunicación hace uso de un puerto que no es 5701, el puerto 5701 es de escucha. Para mantener la conexión con el resto de nodos abre un puerto normal como si fuera un cliente para conectarse al 5701 del otro servidor. Se sabe en las capturas anteriores que puertos son de cliente y cuales de servidor porque al iniciar un cliente podemos ver en los logs cual es su puerto, y lo mismo con los de servidor que hablan con otros servidores. La configuración de los puertos que usa como cliente se explicó en el apartado 6.3.



7.2. Escenario de Defensa 2: Ejecución de Tareas Distribuidas (Executor Service)

Como se comentó uno de los puntos más fuertes de HazelCast es la computación distribuida. Para ello, hace uso de una implementación distribuida de la clase java java.util.concurrent.ExecutorService.

Para la realización de este escenario hemos hecho uso de uno de los ejemplos que ofrece HazelCast al descargar el .zip que lo contiene, el código lo hemos sitiado en el directorio "executing-on-all-members". Para ello, se ha necesitado hacer uso de Maven para construir las clases java correspondientes recogido en el script "configuar_executor_service.sh".

Aclarar que no todos los nodos del clúster de HazelCast pueden participar en la computación distribuida. No solo deben estar configurados para ello, sino que además deben estar programados para ello, implementando ciertas clases java que dan soporte a esto, puede obtenerse más información en [31].

Entonces nos encontramos con que en este escenario no tendremos clientes, todo serán servidores HazelCast unidos al clúster, con la peculiaridad de que unos serán esclavos (reciben tareas para ejecutar) y otros serán maestros (crean tareas y las reparten a los esclavos). Para ejecutarlos tenemos respectivos scripts "start-slave.sh" y "start-master.sh", que son scripts que llaman al ejecutable de HazelCast pasándole el parámetro de configuración del fichero "hazelcast.xml".

Otro aspecto importante es el llamado "quorum" que se utiliza para establecer un número mínimo de nodos del clúster para que este continúe operando (realizando tareas) y crean una especie de sub-clúster con los nodos que operan.

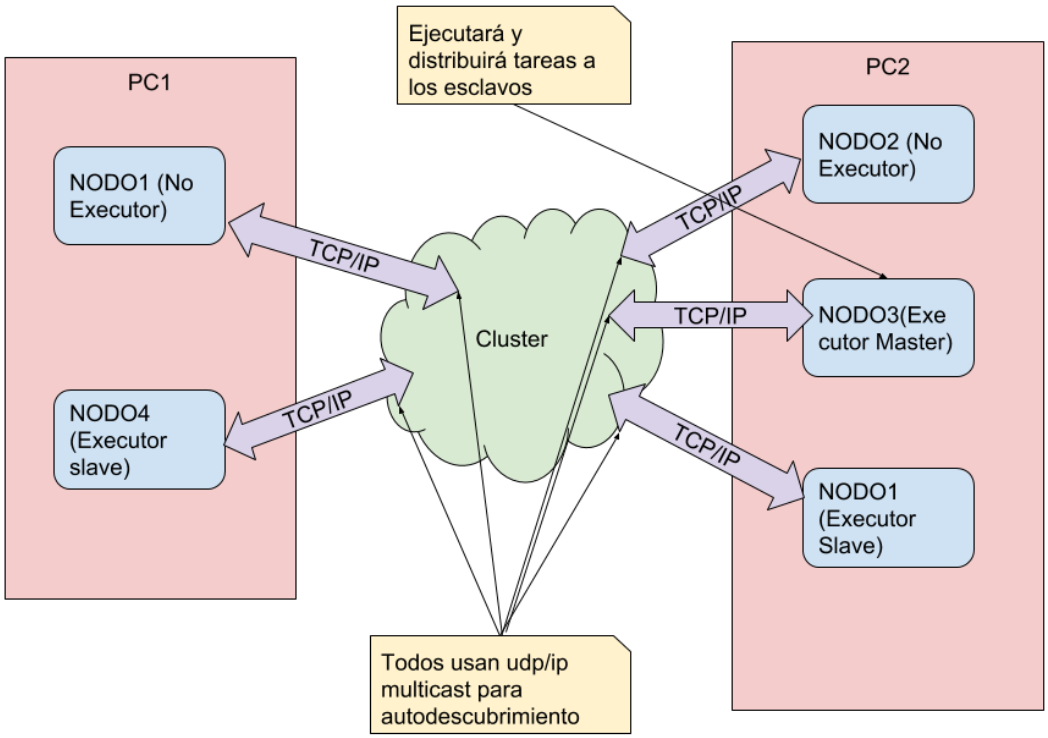
Vamos a realizar las pruebas con 5 miembros en el clúster. Dos corresponden a los ejecutados mediante scripts de servicio en cada equipo, que no realizan tareas. Otro será maestro y otros dos serán esclavos. Se ejecutará el maestro junto a un esclavo en uno de los equipos y el otro esclavo en el otro equipo.

Señalar que los nodos correspondientes a los scripts de servicio no son capaces de realizar tareas, no están programados para ello, por lo que tendremos nodos de los dos tipos en el clúster.

Para ambos escenarios vamos a usar el mismo fichero "hazelcast.xml" ya que las configuraciones son compatibles puesto que los nodos del primer escenario no están programados para ser ejecutores de tareas y por ello ignorarán los apartados relativos a esta configuración que se mostrarán más adelante.

7.2.1. Escenario 2: Esquema de la red

Se presenta un escenario de la red simplificado:



7.2.2. Escenario 2: Configuración del servidor

Para la configuración del servidor es necesario configurar el autodescubrimiento, el puerto de escucha, el grupo, el centro de mantenimiento y las propiedades tal como se vio en el escenario anterior, por lo que no se repetirá.

Además, es necesario configurar el Executor Service. Esto lo realizamos con las siguientes líneas en "hazelcast.xml":

- Para configurar el quorum:

```
<quorum enabled="true" name="miquorum">

      <quorum-size>2</quorum-size>

      <quorum-type>READ_WRITE</quorum-type>

      <probabilistic-quorum             acceptable-heartbeat-pause-
millis="5000" max-sample-size="500" />
```

```
</quorum>
```

EL número mínimo serán 2 nodos, el tipo de este número de nodos mínimo será nodos de escritura y lectura del cluster y probabilistic-quorum es la función que se encarga de determinarl número de nodos que hay de un tipo que puede ser configurado con muchos parámetros, en nuestro caso el tiempo máximo de recepción de la actividad de un nodo antes de asimilar que se ha caído, max-sample-size es el número de muestras con las que se calculará la media y la desviación típica de uniones de nodos al clúster.

- Para configurar el Executor Service:

```
<executor-service name="exec">
      <pool-size>16</pool-size>
      <queue-capacity>10</queue-capacity>
      <statistics-enabled>true</statistics-enabled>
      <quorum-ref>miquorum</quorum-ref>
 </executor-service>
```

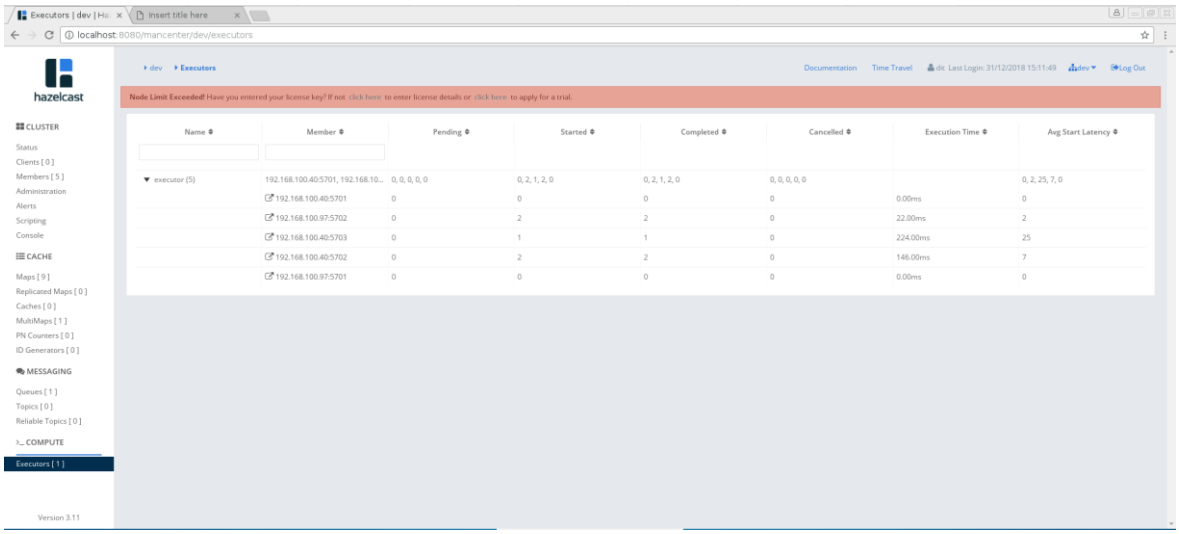
Donde pool-size es el número máximo de hilos de ejecución del nodo, queue-capacity es la capacidad de la cola de entrada de tareas al nodo, statiscs-enable es para activar la capacidad de obtener estadíscticas y quorum-ref debe hacer referencia al quorum anteriormente definido.

Por último, se han modificado los scripts “start-master.sh” y “start-slave.sh” para pasarle el parámetro de configuración de la ruta de “hazelcast.xml”

7.2.3. Escenario 2: Tests y Pruebas del Escenario

Para comprobar que funciona correctamente, en este escenario es necesario hacer uso del centro de mantenimiento, donde se reflejarán los tiempos de ejecución de tareas en cada nodo, el número de tareas resueltas por cada nodo, tareas pendientes en cada nodo, tareas en ejecución en cada nodo, así como la latencia de resolución de tareas.

Se muestra una prueba de esto:



Se amplían a continuación los datos de la misma imagen:

192.168.100.40:5701, 192.168.10...	0, 0, 0, 0, 0	0, 2, 1, 2, 0	0, 2, 1, 2, 0	0, 0, 0, 0, 0		0, 2, 25, 7, 0
192.168.100.40:5701	0	0	0	0	0.00ms	0
192.168.100.97:5702	0	2	2	0	22.00ms	2
192.168.100.40:5703	0	1	1	0	224.00ms	25
192.168.100.40:5702	0	2	2	0	146.00ms	7
192.168.100.97:5701	0	0	0	0	0.00ms	0

Podemos ver que como explicamos antes hay dos nodos que no realizan tareas, correspondientes a los arrancados con los scripts de servicio.

7.2.4. Escenario 2: Análisis del intercambio real de mensajes de red

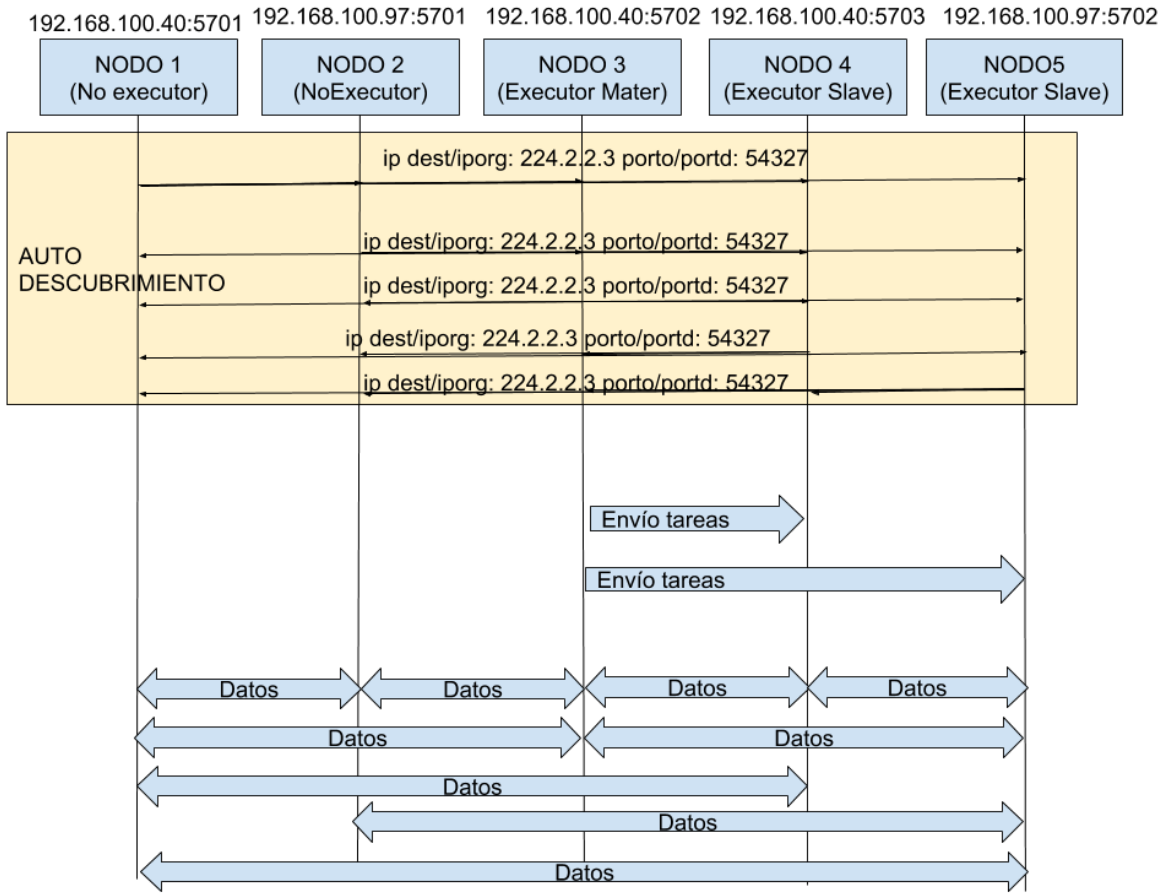
Al igual que en el escenario anterior se pueden ver en Wireshark los mensajes entre nodos, en este caso no hay clientes, se muestra un fragmento de la secuencia de mensajes de intercambio de datos entre algunos nodos. El intercambio de datos en este caso puede ser envío de tareas por parte del maestro a los esclavos y almacenamiento de datos en el clúster como resultado de las tareas:

```
78250 1340.531677 192.168.100.97  192.168.100.40  TCP    68 5701 > 49456 [ACK] Seq=351555 Ack=351591 Win=1022 Len=0 TSval=23198452 TSecr=2354937
78258 1340.717517 192.168.100.40  192.168.100.40  TCP   341 [TCP segment of a reassembled PDU]
78259 1340.717527 192.168.100.40  192.168.100.40  TCP    68 57958 > 5701 [ACK] Seq=355623 Ack=1071020 Win=1022 Len=0 TSval=2355123 TSecr=2355123
78260 1340.717538 192.168.100.40  192.168.100.40  TCP   341 [TCP segment of a reassembled PDU]
78289 1341.542092 192.168.100.97  192.168.100.40  TCP  1792 [TCP segment of a reassembled PDU]
78290 1341.542108 192.168.100.40  192.168.100.97  TCP    68 5701 > 55707 [ACK] Seq=1065106 Ack=353401 Win=1009 Len=0 TSval=2355947 TSecr=23199462
78295 1341.544581 192.168.100.97  192.168.100.40  TCP  1742 [TCP segment of a reassembled PDU]
78296 1341.544587 192.168.100.40  192.168.100.97  TCP    68 5701 > 55707 [ACK] Seq=1065106 Ack=355075 Win=1008 Len=0 TSval=2355950 TSecr=23199464
78301 1341.547315 192.168.100.97  192.168.100.40  TCP   341 [TCP segment of a reassembled PDU]
78302 1341.547321 192.168.100.40  192.168.100.97  TCP    68 5701 > 55707 [ACK] Seq=1065106 Ack=355348 Win=1016 Len=0 TSval=2355952 TSecr=23199467
78307 1341.556390 192.168.100.97  192.168.100.40  TCP   104 [TCP segment of a reassembled PDU]
78308 1341.556393 192.168.100.40  192.168.100.97  TCP    68 5701 > 55707 [ACK] Seq=1065106 Ack=355384 Win=1021 Len=0 TSval=2355962 TSecr=23199476
78329 1342.062356 192.168.100.97  192.168.100.40  TCP   341 [TCP segment of a reassembled PDU]
78330 1342.062419 192.168.100.40  192.168.100.97  TCP    68 5701 > 33843 [ACK] Seq=1068456 Ack=355348 Win=1020 Len=0 TSval=2356467 TSecr=23199981
78333 1342.068122 192.168.100.97  192.168.100.40  TCP   341 [TCP segment of a reassembled PDU]
78334 1342.068149 192.168.100.40  192.168.100.97  TCP    68 49456 > 5701 [ACK] Seq=351591 Ack=351828 Win=1022 Len=0 TSval=2356473 TSecr=23199987
```

Los mensajes de autodescubrimiento son iguales que en el escenario anterior, se muestran a continuación. Señalar que se esta utilizando siempre el descubrimiento multicast porque es el que conlleva la configuración menos direcciones ip ya que si utilizamos el descubrimiento sobre tcp/ip habría que configurar para cada instancia las direcciones ip y puertos del resto de nodos(instancias), lo que probablemente acabaría en algún error, mientras que con multicast solo es necesario configurar una dirección ip y un puerto.

584	31.16805347	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
585	31.16807393	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
586	31.16837629	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
587	31.16866221	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
588	31.16881413	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
589	31.16904606	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
590	31.29091473	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
591	31.29092862	192.168.100.97	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
592	31.29126614	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
593	31.29149030	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
594	31.29165025	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
595	31.29195319	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
5407	64.35169108	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
5408	64.35195367	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
5409	64.35214591	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
5410	64.35242307	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
5424	64.54636339	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
5425	64.54667593	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
5426	64.54708076	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
5427	64.54782119	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
9034	78.50781977	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
9035	78.50807337	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
9036	78.50847600	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
9037	78.50876106	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
9043	78.62459017	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
9044	78.62632687	192.168.100.40	224.2.2.3	UDP	281	Source port: 54327 Destination port: 54327
9045	78.63033546	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327
9046	78.63151013	192.168.100.40	224.2.2.3	UDP	265	Source port: 54327 Destination port: 54327

El diagrama de paso de mensajes en este escenario sería simplemente el autodescubrimiento como en el diagrama del escenario anterior pero con más nodos seguido de un intercambio constante de datos entre los nodos del clúster.



8. Interfaz gráfica de administración del servidor

HazelCast ofrece una aplicación web basada en un archivo .war para la gestión del clúster y los nodos que lo forman. Esta aplicación web se debe descargarse e instalarse por separado y debe configurarse en el servidor de HazelCast (en "hazelcast.xml") la dirección ip, puerto y alias donde se ha instalado para poder comunicarse vía REST. En nuestro caso, esta línea de configuración es la siguiente:

```
<management-center enabled="true" update-interval="3">
http://localhost:8080/mancenter/</management-center>
```

Para ejecutar la aplicación web en nuestro caso hemos creado un script llamado "hazelcast_mancenter.sh" que ejecuta el archivo .war mediante

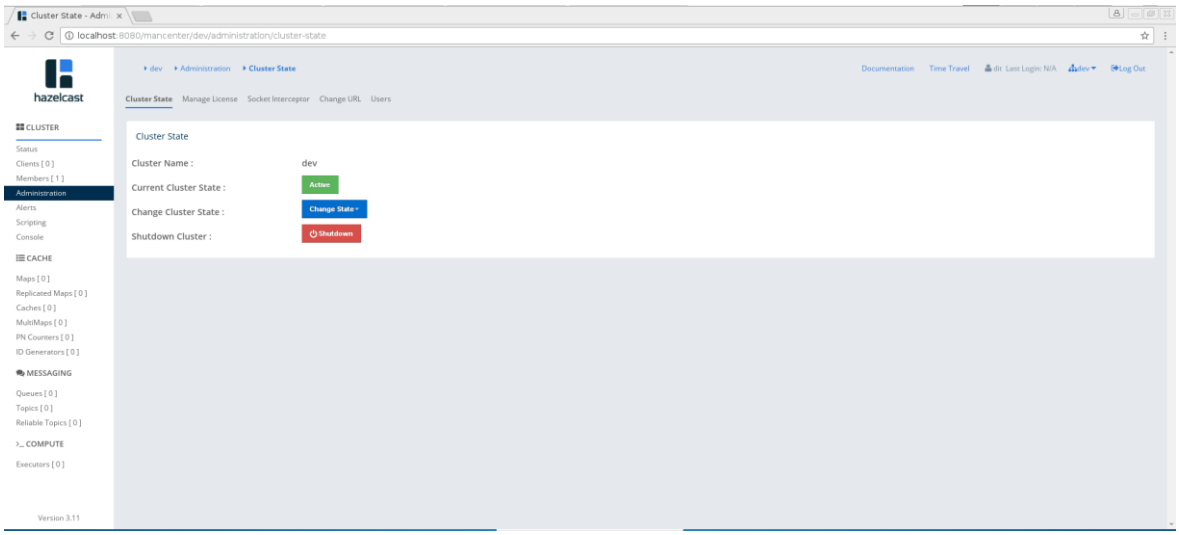
```
java -jar hazelcast-management-center-3.11/hazelcast-mancenter-3.11.war
8080 mancenter
```

Donde 8080 es el puerto donde escuchará y “mancenter” es el alias donde escuchará.

A partir del centro de mantenimiento podemos ver y cambiar el estado de ciertos nodos o de un clúster completo y además recoger estadísticas y generar gráficos con estas. Aunque estas gráficas no podemos obtenerlas ya que es necesario que los clientes envíen los datos y en nuestro caso al no haber creado los clientes y haber usado los de ejemplo estos no envían este tipo de datos al centro de mantenimiento.

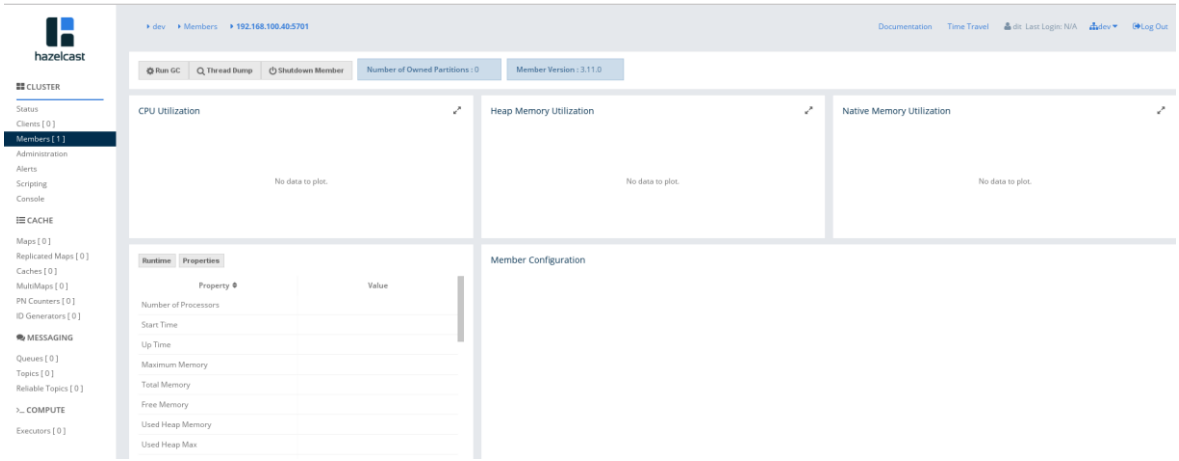
Podemos ver el tamaño de las estructuras de datos almacenadas, el número de executor services, ejecutar scripts en nodos, programar alertas, etc.

A continuación, se muestra una captura de la pantalla de cambio de estado de un clúster:



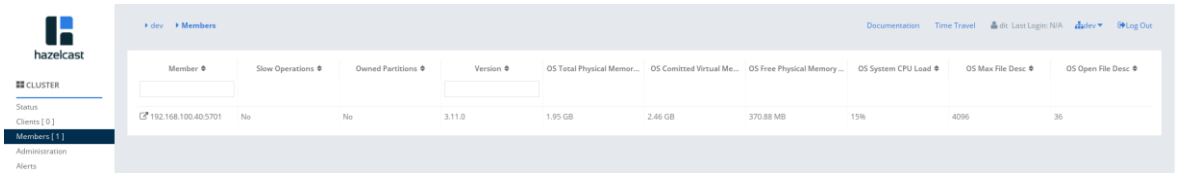
Cuando se pretende cambiar el estado de un clúster no pide un usuario y contraseña que son los configurados en “hazelcast.xml” en el apartado “group” que ya se vio anteriormente.

También podemos ver el número de particiones de un nodo y las estadísticas asociadas a este clickando sobre su dirección ip en el apartado members que muestra todos los nodos conectados al mancenter. Se muestra captura a continuación:



Como se ha comentado el espacio para las gráficas está vacío porque las aplicaciones clientes no están enviando datos.

Y se muestra también una captura de como se ven los miembros, donde se muestra la memoria ocupada, porcentaje de CPU usado , etc.



9. Deficiencias del servicio

La principal deficiencia que se ha encontrado es que al no utilizar la versión de pago sino la versión OpenSource, las características de seguridad no están disponibles.

Por lo demás, se considera que es un sistema bastante completo, es más, con muy poca configuración logra características muy avanzadas lo que justifica su posición en el mercado.

10. Ampliaciones/mejoras del servicio

Como ampliación sobre el funcionamiento básico del servicio, es decir, más allá de ejecutar ejemplos de clientes básicos, se ha realizado la integración con el servidor web Tomcat para el almacenamiento distribuido de sesiones web. Para realizar esta configuración también ha sido necesario utilizar el servidor web Apache como balanceador de carga. Queda constancia de esto en el primer escenario.

11. Incidencias y principales problemas detectados

- Inicialmente, al no haber estudiado nunca nada acerca de la computación distribuida ni sobre este tipo de sistemas me encontraba un poco perdido. Fue necesaria la lectura de bastante información para empezar a comprender para que servía el servicio estudiado, esto en sí no es un problema técnico pero me ha parecido reseñable.

- El primer problema técnico que hubo que solucionar fue que, en la creación de los scripts de servicio el PID del servicio no se almacenaba correctamente puesto que no se ha hecho uso de la función `daemon` y por ello no era posible detenerlo con `service hazelcast stop`, ya que al llamar en el script de servicio a `killproc` el fichero con el pid no existía. Para solucionarlo, en la función start del script de servicio, justo después de llamar al ejecutable, se añadieron las siguientes líneas que almacenaban el pid en el fichero correspondiente para ser encontrado por killproc:

```
$PID=$!

Echo $PID > /var/run/${prog}.pid
```

- Otro problema fue que a la hora de compilar el Openjdk9 este necesitaba 8Gb de espacio en disco libre pero la máquina virtual solo contaba con 6GB, por lo que fue necesario añadirle un nuevo disco y montarlo. Por ello todo el trabajo se ha realizado sobre una carpeta montada sobre un disco secundario en el que se creó una única partición con formato ext4 mediante el comando:

```
fdisk /deb/sdb

mkfs.ext4 /dev/sdb1
```

```
mount -t /dev/sdb1 /home/dit/Trabajo
```

Por último, se modificó el fichero /etc/fstab añadiendo la línea:

```
/dev/sdb1          /home/dit/Trabajo       ext4        defaults     0
      0
```

para que siempre se montara el nuevo disco con el arranque del sistema.

- Hubo un problema que trajo consigo grandes quebraderos de cabeza, este fue que al integrar HazelCast con Tomcat no se habían asignado los permisos necesarios al ejecutable de HazelCast para que pudiese ser ejecutado con el servidor Tomcat. Este problema que parece muy simple de resolver no fue para nada fácil de detectar puesto que los servidores no se iniciaban, pero no aparecía ningún tipo de log que diese una pista de porqué.

- Al hacer uso de `service hazelcast start` el texto que imprimía el servidor por la salida estándar y la salida de errores no podía verse, por tanto, no era posible ver porque fallaba o simplemente si se ejecutaba correctamente. Para solucionar esto, se me ocurrió redireccionar la salida estándar y la salida de errores del comando que ejecuta el servidor hacia un fichero en /var/log/hazelcast y con esto conseguí solucionar el problema.

- Otro punto problemático ha sido entender que hacía cada ejemplo de código ya que estos se proporcionan sin ningún tipo de explicación o de readme, ni existe una página en la referencia que lo explique. Por ello, me costó entender el funcionamiento y como configurar el escenario del Executor Service, tuve que ir probando varias cosas que se me iban ocurriendo hasta que lo entendí.

12. Resumen y Conclusiones

Como se ha podido observar HazelCast tiene un abanico de funcionalidades muy amplio y se han realizado escenarios de dos de los más relevantes.

HazelCast proporciona un nivel de configuración mucho más profundo del realizado, pero que para los escenarios creados no ha sido necesario utilizar, sin embargo, se deja constancia de ello en el Anexo A.

Hemos observado que HazelCast es uno de los líderes en el mercado de los IMDG y que realiza tareas más allá del funcionamiento básico de un IMDG.

La teoría en la que se fundamenta es compleja, pero han conseguido crear un servicio que con una configuración simple cuando funciona solo y media para integrarlo con otros servidores ofrece unas funcionalidades sorprendentes.

Y un aspecto importante es su característica opensource, aunque ya se ha visto que con la versión gratuita podemos tener problemas de seguridad.

ANEXO A: Parámetros de configuración y comandos de gestión del servidor

Aunque durante los apartados anteriores se han comentado algunos parámetros de configuración, se van a comentar ahora los diferentes parámetros de configuración del fichero “hazelcast.xml”.

Debido a la cantidad de opciones de configuración no se van a describir todas las opciones ni nos vamos a centrar en que significa cada parámetro de los expuestos. Tampoco se incluirán opciones de la versión de pago. Pueden consultarse todas las opciones de forma detallada en [32]. Simplemente se dejarán ejemplos de configuración sacados de la referencia con una breve descripción de para que se usa.

Empezamos por la directiva “import” que nos permite cargar otros archivos xml con configuración, lo que sirve, por ejemplo, para separar estos según funcionalidad.

```
<import resource="your-configuration-file.xml"/>
```

Tenemos también la directiva <config-replacers> que nos permite reemplazar variables por contenido, por ejemplo, para introducir contraseñas.

```
<config-replacers fail-if-value-missing="false">
    <replacer class-name="com.hazelcast.config.replacer.EncryptionReplacer">
    <properties>
            <property name="passwordFile">password.txt</property>
            <property name="passwordUserProperties">false</property>
            <property name="cipherAlgorithm">DES</property>
            <property name="keyLengthBits">64</property>
            <property name="secretKeyAlgorithm">DES</property>
            <property
name="secretKeyFactoryAlgorithm">PBKDF2WithHmacSHA1</property>
        </properties>
   </replacer>
</config-replacers>
```

La directiva <group> que ha sido comentada para espeficicar el grupo y la contraseña:

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

La directiva <instance-name> que nos permite poner un nombre a una instancia de HazelCast para obtenerla (desde el código a través de la API):

```
<instance-name>hzInstance1</instance-name>
```

Para configurar el centro de mantenimiento, como ya se comentó se utiliza:

```
<management-center enabled="true" scripting-enabled="false" update-interval="2">http://localhost:8080/hazelcast-mancenter</management-center>
```

Donde scripting-enabled nos permite ejecutar scripts en este y update-interval es el periodo de actualización.

Tenemos propiedades, que nos permiten incorporar algún módulo y configurarlo, como por ejemplo ssl, y deben ir dentro de las directivas:

```
<properties>
    <property name="your-property">Value of the property</property>
</properties>
```

Dentro de la directiva <properties> pueden situarse los siguientes módulos, aunque la mayoría están disponibles solo para la versión de pago:

- <discovery-strategy>
- <map-store>
- <queue-store>
- <wan-replication>
- <ssl>
- <service>
- <login-module>
- <security-object>

- <socket-interceptor>

También se deben configurar los mecanismos de autodescubrimiento, que ya han sido comentados en otros apartados de esta memoria y no entraremos en los mecanismos para descubrimiento en entornos cloud.

Existen directivas para configurar mecanismos de encriptación de sockets, ssl, etc. Pero solo están disponibles para la versión de pago. Se deja un ejemplo de ello:

```
<symmetric-encryption enabled="true">
      <algorithm>PBEWithMD5AndDES</algorithm>
      <salt>thesalt</salt>
      <password>thepass</password>
      <iteration-count>19</iteration-count>
</symmetric-encryption>
```

Pasamos a las directivas relativas a la configuración de red, donde omitiremos las relativas a cloud y a características que solo están disponibles en la versión de pago. Se presenta un ejemplo:

```
<network>
      <public-address>11.22.33.44:5555</public-address>
     <port auto-increment="true" port-count="100">5701</port>
     <outbound-ports>
          <ports>34500</ports>
     </outbound-ports>
     <reuse-address>false</reuse-address>
     <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>
         </tcp-ip>
```

```
</join>
<interfaces enabled="true">
       <interface>10.10.1.*</interface>
</interfaces>
```

El proveedor de direcciones es principalmente necesario también para entornos cloud pero dejamos un ejemplo:

```
<member-address-provider enabled="false">
       <class-name>com.hazelcast.MemberAddressProviderImpl</class-name>
       <properties>
            <property name="prop1">prop1-value</property>
            <property name="prop2">prop2-value</property>
      </properties>
</member-address-provider>
```

Por último en cuanto configuración de red, tenemos el failure detector, que configura ciertos parámetros relativos a detectar cuando uno de los nodos ha caído:

```
<failure-detector>
    <icmp enabled="true">
            <timeout-milliseconds>1000</timeout-milliseconds>
           <fail-fast-on-startup>true</fail-fast-on-startup>
            <interval-milliseconds>1000</interval-milliseconds>
            <max-attempts>2</max-attempts>
            <parallel-mode>true</parallel-mode>
           <ttl>255</ttl>
    </icmp>
</failure-detector>
</network>
```

Tenemos también la directiva <partition-group> que nos permite especificar los miembros que forman cada grupo de particiones, puesto que podemos tener varios grupos:

```
<partition-group enabled="true" group-type="CUSTOM">
      <member-group>
          <interface>10.10.0.*</interface>
          <interface>10.10.3.*</interface>
```

```
            <interface>10.10.5.*</interface>
        </member-group>
        <member-group>
            <interface>10.10.10.10-100</interface>
            <interface>10.10.1.*</interface>
            <interface>10.10.2.*</interface>
         </member-group>
</partition-group>
```

La configuración del executor service ya fue explicada en el apartado relativo al segundo escenario. Existe otro tipo de executor service que es el durable executor service, cuya diferencia es que este no dura para siempre sino que se configura el tiempo de duración del mismo, que se muestra a continuación:

```
<durable-executor-service name="default">
      <pool-size>16</pool-size>
      <durability>1</durability>
      <capacity>100</capacity>
     <quorum-ref>quorumRuleWithThreeNodes</quorum-ref>
</durable-executor-service>
```

Y una tercera versión de este es el scheduled executor service que se configura como se muestra:

```
<scheduled-executor-service name="default">
      <pool-size>16</pool-size>
     <durability>1</durability>
     <capacity>100</capacity>
     <quorum-ref>quorumRuleWithThreeNodes</quorum-ref>
     <merge-policy batch-size="100">PutIfAbsentMergePolicy</merge-policy>
</scheduled-executor-service>
```

Es posible también configurar cad tipo de estructura de datos que se almacena en el clúster como pueden ser mapas, colas, multimapas, mapas replicados, semaforos, el uso como

caché, listas, etc. La cantidad de estos elementos es muy amplia por lo se va a mostrar solo el ejemplo de la cola:

```
<queue name="default">
   <statistics-enabled>true</statistics-enabled>
   <max-size>0</max-size>
   <backup-count>1</backup-count>
   <async-backup-count>0</async-backup-count>
   <empty-queue-ttl>-1</empty-queue-ttl>
   <item-listeners>
   <item-listener include-
value="true">com.hazelcast.examples.ItemListener</item-listener>
   </item-listeners>
   <queue-store>
      <class-name>com.hazelcast.QueueStoreImpl</class-name>
      <properties>
         <property name="binary">false</property>
         <property name="memory-limit">1000</property>
         <property name="bulk-load">500</property>
      </properties>
   </queue-store>
   <quorum-ref>quorumRuleWithThreeNodes</quorum-ref>
   <merge-policy batch-size="100">PutIfAbsentMergePolicy</merge-policy>
</queue>
```

Podemos configurar la creación de Listeners a partir de implementar nosotros mismos las interfaces de java MembershipListener, DistributedObjectListener, MigrationListener y PartitionLostListener. Y haciendo uso en el fichero de configuración de:

```
<listeners>
     <listener>your-package.YourMembershipListener</listener>
     <listener>your-package.YourDistributedObjectListener</listener>
     <listener>your-package.YourMigrationListener</listener>
     <listener>your-package.YourPartitionLostListener</listener>
</listeners>
```

Donde your-package.your… es la clase que hemos implementado.

Podemos configurar como hazelcast hace uso de la memoria mediante las siguientes directivas:

```
<native-memory allocator-type="POOLED" enabled="true">
    <size unit="MEGABYTES" value="256"/>
    <min-block-size>32</min-block-size>
    <page-size>4194304</page-size>
    <metadata-space-percentage>12.5</metadata-space-percentage>
</native-memory>
```

Para configurar el despliegue de código que hayamos creado en el clúster es necesario hacer uso de las siguientes directivas:

```
<user-code-deployment enabled="true">
   <class-cache-mode>ETERNAL</class-cache-mode>
   <provider-mode>LOCAL_CLASSES_ONLY</provider-mode>
   <blacklist-prefixes>com.foo</blacklist-prefixes>
   <whitelist-prefixes>com.bar.MyClass</whitelist-prefixes>
   <provider-filter>HAS_ATTRIBUTE:lite</provider-filter>
</user-code-deployment>
```

Como ya hemos dicho existen más opciones de configuración que no se han nombrado ya sea porque pertenecen a la versión de pago o porque no se consideran relevantes y esta memoria se alargaría demasiado. Puede consultarse la referencia completa del archivo de configuración en [32].

ANEXO B: Parámetros de configuración y comandos de gestión del cliente

No aplica porque no existe un cliente concreto para el servicio, cada usuario debe crear el suyo.