UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

MÁSTER EN INGENIERÍA DE TELECOMUNICACIONES

Trabajo Fin de Máster

"Desarrollo de un sistema de autoescalado dinámico de base de datos distribuida MongoDB sobre una plataforma cloud OpenStack"

Autor: María Bermejo Corrales
Tutor: Juan A. Gómez Pulido
Cotutor: Miguel A. Díaz Corchero

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

MÁSTER EN INGENIERÍA DE TELECOMUNICACIONES

Trabajo Fin de Máster

"Desarrollo de un sistema de autoescalado dinámico de base de datos distribuida MongoDB sobre una plataforma cloud OpenStack"

Autor:

Fdo.:

Directores:

Fdo.:

Tribunal Calificador:

Presidente:

Fdo.:

Vocal:

Fdo.:

Secretario:

Fdo.:

Agradecimiento:

El presente trabajo se ha podido llevar a cabo gracias al Centro Extremeño de Tecnologías Avanzadas (CETA-CIEMAT), el cual me ha dado la oportunidad de desarrollar el trabajo ofreciendo tanto los recursos necesarios como todo el conocimiento que ha podido aportar el equipo de "Unidad de Sistemas y Explotación". Gracias por acogerme en el centro y sentirme como una más. Agradecer en especial a Miguel A. Díaz Corchero su apoyo y ayuda diaria.

"Este trabajo se ha llevado a cabo haciendo uso de la infraestructura de computación facilitada por el Centro Extremeño de Tecnologías Avanzadas (CETA-CIEMAT), financiado por el Fondo Europeo de Desarrollo Regional (FEDER). El CETA-CIEMAT pertenece al CIEMAT y al Gobierno de España"

Dedicatoria:

Trabajo dedicado especialmente a quienes me han apoyado en el duro esfuerzo de compaginar trabajo y estudios en un momento de recién incorporación al mundo laboral. Mi familia que gracias a ellos comencé este camino con el incondicional apoyo moral y sentimental. Mi pareja gracias a la cual consigo paz, sosiego y nunca fallido cariño.

ÍNDICE:

INDICE ILUSTRACIONES:

ÍNDICE TABLAS:

ABSTRACT

This project involves the implementation of a process that extends a database considering the storing of it and the resources being used, to avoid the saturation that may occur, the stop services or malfunctions, and to allow performing critical tasks on a set time. This expansion is carried out using computing technology in the cloud which facilitates this task.

Computing platform in the cloud is OpenStack, which provides easily resources for the user through a web interface, with the ability to manage them once are created. Besides making use of this control panel in the project to assemble the entire infrastructure of the database, we used the console commands for the creation of the automation process.

The database is MongoDB, and is provisioned from zero to reach a mature production environment to work in a situation as real as possible, simulating problems that the administrators can find, focusing on filling storage. One advantage of MongoDB is its ease of scale-out storage using a method that delivers the data by different servers that we have configured in infrastructure, according to the method of Sharding. These data will be continuously monitored, controlling the growth among between different servers to the stipulated time, expand resources, etc.

1. RESUMEN

Este proyecto consiste en la realización de un proceso que amplía una base de datos teniendo en cuenta el almacenamiento de la misma y los recursos que está utilizando, para evitar su saturación y que se puedan producir paradas de servicio o mal funcionamiento, así como para permitir la realización de tareas críticas en un tiempo ajustado. Esta ampliación se lleva a cabo haciendo uso de una tecnología de computación en la nube que facilita dicha tarea.

La plataforma de computación en la nube es Openstack, que provisiona recursos de manera sencilla para el usuario desde una interfaz web, con la posibilidad de administrarlos una vez creados. Además de hacer uso de este panel de control en el desarrollo del proyecto para montar toda la infraestructura de la base de datos, hemos utilizado los comandos de consola para la creación del proceso de automatización.

La base de datos es MongoDB, y se provisiona desde cero hasta llegar a un entorno maduro de producción para trabajar en una situación lo más real posible, simulando los problemas que se encuentran los administradores, haciendo foco al llenado de almacenamiento. Una de las ventajas de MongoDB es su facilidad de escalado horizontal empleando un método de almacenaje que reparte los datos por los diferentes servidores que tengamos configurados en la infraestructura, según el método de Sharding. Estos datos estarán continuamente monitorizados, controlando el crecimiento entre los diferentes servidores para el momento que se estipule, ampliar recursos, etc.

Palabras Clave:

Cloud Computing, Openstack, Interfaz Web, Dashboard, MongoDB, Escalado horizontal y vertical, Máquina Virtual - Instancia, Infraestructura como Servicio (IaaS), Software como Servicio (SaaS), Plataforma como Servicio (PaaS), “on premises” / “off premises”, Hipervisor, NoSQL, Big Data, Base de dato o DB, administrator de bases de datos (DBA), Background o Segundo plano, Script, *Python Django,* Framework, Documento y Colección, MMS, almacenamiento o Storage, Proceso asíncrono, Miembros Primario / Secundarios, Failover o tolerancia a fallos, Nodo Standalone, Chunks o trozos, Clave de shard, Cliente Nova.

2. INTRODUCCIÓN

En este proyecto se ha hecho uso de dos tecnologías de software libre y código abierto (denominación Open Source) actualmente en auge, con un grado de respaldo y aceptación bastante amplio por la comunidad de desarrolladores, administradores y usuarios de dichas tecnologías.

Openstack es una plataforma (*bajo licencia Apache 2.0.*) de computación en la nube o Cloud Computing, que proporciona infraestructura como servicio (Infraestructure as a Service, IaaS) para gestionar entornos públicos y privados o híbridos desde una misma interfaz, llamada interfaz web. La tecnología consiste en varios proyectos relacionados que se encargan del procesamiento, almacenamiento y gestión de red.

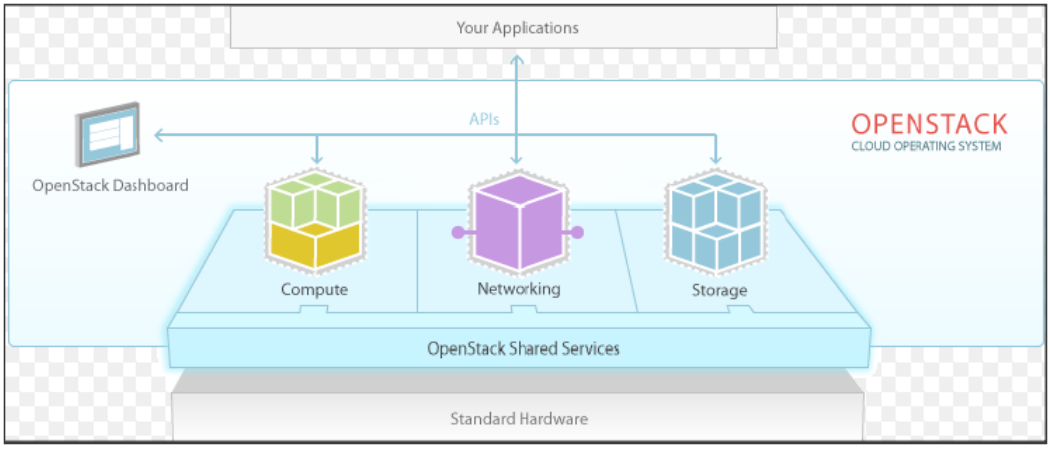


Ilustración 1: Arquitectura Openstack.

La computación en la nube o Cloud Computing (según la definición del NIST – National Institute of Standars and Technology) es "*un modelo para habilitar acceso conveniente por demanda a un conjunto compartido de recursos computacionales configurables (redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente aprovisionados y liberados con un esfuerzo mínimo de administración o de interacción con el proveedor de servicios*" [1]. Este modelo de nube promueve la disponibilidad y está compuesto por cinco características esenciales, tres modelos de servicio y cuatro modelos de despliegue.

Algunas de las características del Cloud Computing son las siguientes [2]:

- ✓ Servicio a demanda: Los recursos se ofrecen como servicio a los usuarios.
- ✓ Accesible a través de la red.

- ✓ Agrupación de recursos en diferentes grupos: Almacenamiento, memoria RAM, cores, etc.
- ✓ Independencia de recursos: Capacidad de aislar tráfico, datos o configuraciones de usuarios utilizando el mismo software.
- ✓ Escalabilidad y Elasticidad: Permite ajustar los recursos utilizados a demanda.
- ✓ Pago por uso: Permite ajustar los costes de explotación al uso de los recursos.

Las soluciones de Cloud Computing existentes se clasifican según diferentes criterios, que fundamentalmente son: modelos de servicio y formas de implementación.

Modelos de servicio:

- SaaS – Software as a Service: Se entregan al cliente aplicaciones como servicio. Se ofrecen licencias de aplicaciones para su uso bajo demanda.
- PaaS – Platform as a Service: Esta capacidad le permite al consumidor desplegar en la infraestructura del proveedor aplicaciones creadas por el consumidor o adquiridas, usando lenguajes de programación y herramientas del proveedor.
- IaaS – Infrastructure as a Service: Se pone a disposición del usuario un pool (flota) de recursos (cpu, memoria, disco...) como servicio. El usuario busca evitar la inversión en Hardware, y gastos asociados a DataCenters propios o en alquiler.

Formas de implementación:

- Nube privada: La infraestructura de esta nube es operada únicamente para una organización. Puede ser administrada por la organización o por un tercero y puede existir dentro de la misma (on premises) o fuera de la misma (off premises).
- Nube comunitaria: La infraestructura de esta nube es compartida por varias organizaciones y apoya las preocupaciones de una comunidad particular sobre un tema específico, por ejemplo: seguridad, investigación, políticas, etc. Puede ser administrada por la organización o por un tercero y, al igual que la nube privada, puede existir on premises u off premises.
- Nube pública: La infraestructura de esta nube está disponible para el público en general o para un gran grupo, y dicha infraestructura la proporciona una organización que vende servicios en la nube.

- Nube híbrida: Es la composición de dos o más nubes, por ejemplo privada y pública, que permanecen como entidades únicas pero que coexisten por tener tecnología que permite compartir datos o aplicaciones entre las mismas. Es el caso de un escenario donde la aplicación se desarrolla y se prueba en una nube privada y luego se despliega a una nube pública.

¿Qué motiva a las empresas a utilizar Openstack? El reducido tiempo de despliegue de servicios, la reducción de los costes operativos de TI y proporcionar una infraestructura de TI más receptiva (según Red Hat).

La receptividad viene originada por la comunicación de cada proyecto vía API (Rest). La existencia de estos API propicia la programación de aplicaciones externas con todo tipo de propósitos, ya que la comunicación con cualquier componente de OpenStack es fácil y rápida.

> [3] *"Openstack: plataforma que todos podemos usar para crear nubes abiertas enormemente escalables".*

El proyecto de *Openstack* está gestionado por la Fundación *OpenStack* que promueve el software *OpenStack* y su comunidad. En la siguiente dirección pueden verse todas las empresas de la comunidad *Openstack*: miembros platino, oro, empresas colaboradoras y organizaciones de apoyo que ayudan a *Openstack* a seguir desarrollándose. Está basado como puede verse en un modelo que recibe colaboraciones de todos haciendo el software mucho más rico e innovador constantemente.

https://www.openstack.org/foundation/companies/

Ilustración 2: Miembros platino de la comunidad de Openstack.

A mediados de 2010, Rackspace y NASA anunciaban de manera conjunta que estaban desarrollando un nuevo software IaaS para solucionar los problemas que NASA había encontrado en el desarrollo de sus propias soluciones, y para convertirse en una alternativa real a Amazon Web Services. Así nació OpenStack como una

alternativa completamente abierta al amparo de gran parte de la industria al dominio de Amazon en la Nube Pública y de VMware en la Nube Privada.

Aunque comenzó como una iniciativa conjunta de la NASA y Rackspace y estaba orientada solo hacia la virtualización de Linux, OpenStack ahora se ha convertido en una nube de infraestructura como servicios que pueden desplegar instancias en cualquier tipo de hipervisor, incluyendo ESXi de VMware y Windows Hyper-V.

Ahora VMware ha respondido con el lanzamiento de OpenStack integrado con VMware (VIO), con el único propósito de desarrollar una solución en la nube que funcione sobre una infraestructura de vSphere (disponible como un complemento gratuito) con un asistente para la configuración ya que uno de los puntos flacos de Openstack es su engorroso procedimiento de puesta en marcha.

Esto nos da una idea del posicionamiento de Opentack en el mercado.

> [4] *"OpenStack seguirá creciendo, y yo podría incluso ir tan lejos como para decir que se convertirá en el estándar de facto para la infraestructura de nube privada. Todo el mundo está en el carro de OpenStack, y están buscando hacer tanto dinero como puedan en el camino"*

En este proyecto Opentack aporta la herramienta para manejar el entorno privado sin necesidad de utilizar procesos manuales por consola tradicionales. Esto nos ha llevado a poder provisionar y des-provisionar recursos, y poder montar fácilmente la infraestructura de base de datos.

Ponemos foco ahora a la Base de Datos, **MongoDB.** Se trata de una base de datos creada por *10gen,* la compañía de MongoDB, con licencia GNU AGPL 3.0. Es una base de datos orientada a documentos, es decir, puede tener un esquema de datos diferente que nada tenga que ver con el resto de registros almacenados.

MongoDB se ha vuelto quizás el motor de base de datos más utilizado para aplicaciones NoSQL. Ofrece un alto rendimiento para las búsquedas, así como la escalabilidad de manera horizontal, agregación en tiempo real, características de datos geoespaciales, etc.

[5] NoSQL (del inglés '*Not Only SQL*') es una filosofía de sistemas de gestión de bases de datos que modifican por completo el modelo clásico de bases de datos relacionales. Las características comunes entre las implementaciones de bases de datos distribuidas no relacionales o NoSQL son las siguientes:

NoSQL	
Consistencia	Los datos no saben de un esquema a priori.
Modelo transaccional	N x inserciones. No es necesario transcribir cada sentencia para poder ser ejecutada.
Escalabilidad Horizontal	Sistema que escala horizontalmente, no verticalmente, con solo añadir más nodos.
Computación	Servidores comunes, no requieren apenas de recursos de computación.
Estructura distribuida	Inserciones de grandes cantidades con estructuras híbridas: tablas HASH.

Tabla 1: Características de Base de Datos NoSQL

El modelo tradicional se aleja de la realidad actual de la llamada "*era de la información",* en la cual la sociedad, los clientes y las empresas generan grandes cantidades de datos que necesitan ser almacenados y procesados, surgiendo términos nuevos como Big Data para esta generación masiva de datos y NoSQL para las bases de datos que están enfocadas a albergarlos y en algunos casos, a procesarlos. *"MongoDB fue construido para la nube" según sus desarrolladores [3].*

Big Data es el conjunto de datos que superan la capacidad de captura, gestión y procesamiento en un tiempo determinado, saturando así la capacidad del software habitual. Además de la captura, gestión y procesamiento, las dificultades más comunes son también la búsqueda, la compartición, la visualización, el almacenamiento y el análisis entre otras. Los dos conceptos importantes de Big data son: Almacenamiento y Procesamiento. Ya no solo tenemos información estructurada en Bases de Datos o Archivos, ahora empezamos a tener información de diferentes tipos y totalmente desestructurada. La velocidad a la que se genera esta información hace imposible gestionarla con sistemas de base de datos convencionales.

¿Qué conclusiones podemos sacar de los conceptos de NoSQL y BigData? Actualmente la mayoría de organizaciones sufren problemas de escalabilidad y rendimiento debido a la cantidad de datos almacenados y la capacidad de computación necesaria para procesar dichos datos. Estos datos, donde se encuentren albergados, van a ser atacados por miles de usuarios concurrentes y con millones de consultas diarias. Se hace entonces necesario utilizar tecnologías que lo resuelvan.

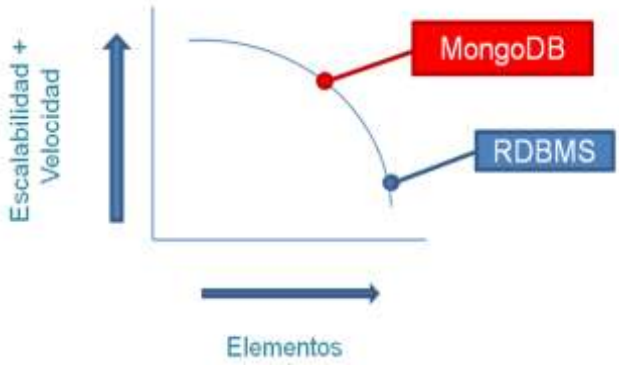


Ilustración 3: MongoDB vs RDBMS

Si utilizamos MongoDB la escalabilidad ligada a la velocidad, será mejor y mayor que las DDBB tradicionales estructuradas.

Why use mongoDB?

¿Y por qué mongo y no otra base de datos NoSQL? En nuestro caso, por el método Sharding y escalado horizontal, y por más razones anteriormente comentadas. Sin embargo, el mercado tiene otras referencias como la prestigiosa consultora de tecnologías de la información *Gartner Inc.*, que la posiciona Mongo como la única base de datos "Challenger" es decir, base de datos Aspirantes o Desafiantes que ofrece productos con buenas características y con un número considerable de instalaciones, aunque sin la visión de los líderes.

Ilustración 4: Gartner Octubre 2014 sobre DataBases.

3. OBJETIVOS

El fin último de este proyecto es aportar la **automatización de una tarea a aquellos usuarios que trabajen con Openstack y MongoDB**. Estas son unas tecnologías actualmente muy usadas en el ámbito empresarial, para la gestión de Clouds y para el almacenamiento y procesamiento de datos que se generen.

En el camino hasta el objetivo final, se lleva a cabo la instalación y administración de estas tecnologías, documentando tanto la parte teórica como práctica y sirviendo cada una de ellas para usuarios que monten desde cero una plataforma similar.

Entre las competencias de un administrador de base de datos, DBA, están las siguientes:

- Diseñar, desplegar y monitorizar servidores de bases de datos.
- Diseñar la distribución de los datos y las soluciones de almacenamiento.
- Previsión de soluciones de crecimiento.

Con esta automatización podremos descuidar la previsión de soluciones adaptadas al crecimiento, en un futuro próximo y tener un plan de acción para cuando llegue el momento. La base de datos estará monitorizada con un proceso en background o segundo plano, y en el momento que se detecte que el almacenamiento provisionado está llegando a un x% del total que establezcamos, comienza la provisión de más recursos, añadiendo los mismos a un cluster que está ya configurado y que alberga los servidores de Bases de Datos en ese momento.

Este clúster está compuesto por servidores que pueden estar en el mismo emplazamiento, distribuido geográficamente, en cloud privadas, en cloud públicas, ya que la plataforma Cloud Computing Openstack, abstrae dicha capa centralizando en un punto la provisión y administración de dichos servidores.[5]

El aporte de conocimiento se basa en la entrega de un script que correrá en uno de los servidores del cluster de la Base de datos. Este servidor, según la arquitectura MongoDB montada, sería conveniente que fuese un servidor con configuración de router que veremos en detalle en capítulos posteriores. El router en el método de Sharding (método de distribución de datos por diversos servidores) es el que decide en qué servidor se alojará cada documento insertado. Desde éste se puede comprobar el estado de toda la arquitectura.

4. FUNDAMENTOS TEÓRICOS Y DESARROLLO

Para la realización del proyecto hace falta el aporte de conocimientos teóricos que se reflejarán tal y como han hecho falta en el desarrollo del mismo.

Comenzamos con una pequeña pincelada de Horizon, la interfaz de Openstack, tras una descripción de carácter introductorio de esta plataforma Cloud Computing. Seguimos con toda la documentación sobre MongoDB, introducción y métodos de replicación y sharding. Y para finalizar, nos centraremos en la parte de Openstack que crea instancias - NOVA -, única parte del proyecto utilizada y estudiada, ya que es objetivo del proyecto el conocer la configuración de la plataforma al completo, así como la administración en profundidad de la misma.

4.1 INTRODUCCIÓN OPENSTACK

Openstack es un software *Open Source* usado para la *construcción* de Clouds. Es una colección de proyectos de software, a través de los cuales Openstack proporciona una completa plataforma operativa para la *administración y gestión* de Clouds.

La misión principal de Openstack es proporcionar el despliegue de una infraestructura IaaS de forma sencilla, escalable, elástica y de cualquier tamaño (tanto Clouds públicos, privados o híbridos aunque no ha sido objeto de este proyecto).

A continuación vamos a mostrar dos esquemas de la conexión de los proyectos de Openstack. El primero nos da una visión más general de cada uno de ellos comparado con el segundo, que es ya una visión detallada con los subproyectos que engloba cada uno de ellos.

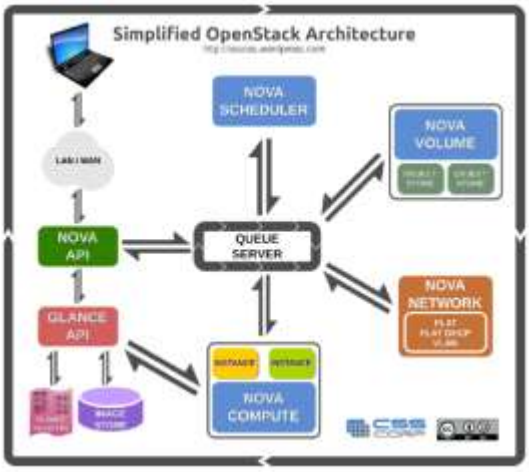


Ilustración 5: Esquema general interconexión proyectos Openstack.

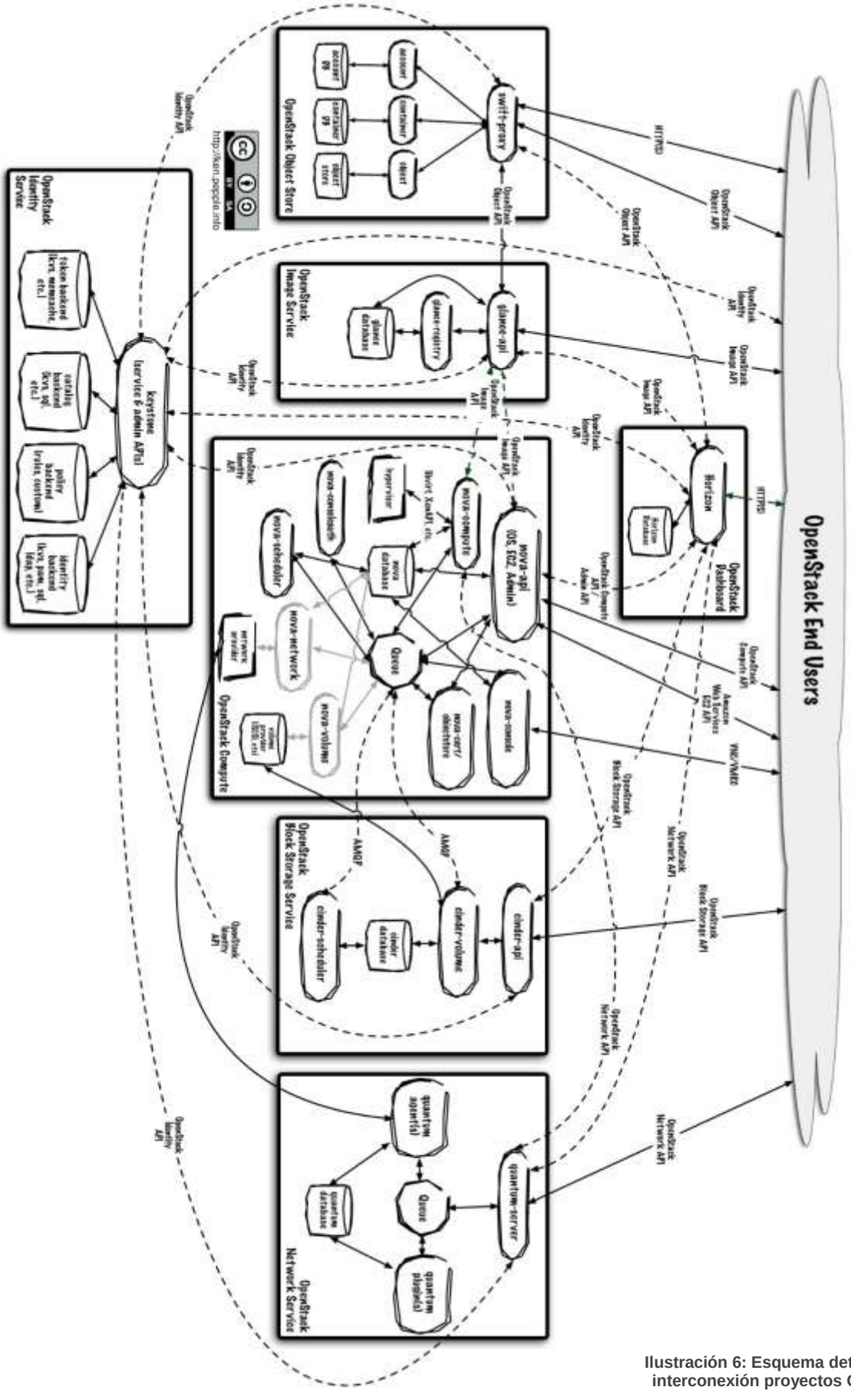


Ilustración 6: Esquema detallado de la interconexión proyectos Openstack.

Servicio	Proyecto	Descripción
Dashboard	**Horizon**	Proporciona una interfaz de usuario modular, basada web, para la gestión de todos los servicios de OpenStack.
	Nova	Proporciona máquinas virtuales bajo demanda. Almacena y recupera imágenes de discos virtuales y sus datos asociados y lanza las instancias.
	Neutron	Permite la conectividad de la red entre un servicio y otro OpenStack. Proporciona una API para que los usuarios puedan definir las redes y los archivos adjuntos en ellos.
Almacenamiento de Objetos	**Swift**	Proporciona almacenamiento de objetos, nos permite almacenar y/o recuperar ficheros.
Almacenamiento de Bloques	**Cinder**	Cinder proporciona una infraestructura para la gestión de volúmenes en OpenStack. En su origen fue un componente llamado nova-volumen, pero se ha convertido en un proyecto independiente. Además realizada copias de seguridad.
Identidad	**Keystone**	Proporciona servicios de autenticación de usuarios y autorización a todos los servicios de OpenStack.
Imágenes	**Glance**	Proporciona un catálogo y un repositorio de imágenes de discos virtuales. (Se puede hacer a través de Glance).
Telemetría	**Ceilometer**	Monitoriza y mide la el consumo de recursos del cloud para su posterior evaluación o para una facturación.
Orquestación	**Heat**	Organiza múltiples aplicaciones en la nube compuesta utilizando cualquiera el formato de la plantilla CALIENTE nativo o la AWS CloudFormation formato de la plantilla, tanto a través de una API REST-OpenStack nativa y un API Query compatible con CloudFormation.
Servicios de Data Base	**Trove**	Proporciona servicios de bases de datos escalables como servicios permite tanto bases de datos relacionales como no relacionales.

Tabla 2: Servicio, Proyecto y Descripción Openstack.

4.2 HORIZÓN, OPENSTACK

Es un panel web para el manejo de instancias y volúmenes y conexiones de red. Este servicio es una aplicación web modular desarrollada con el framework de Python Django que permite comunicarse con las diferentes APIs de Openstack de una forma sencilla. Openstack Dashboard es fundamental para usuarios noveles y en general para realizar acciones sencillas sobre las instancias.

Se muestra a continuación la pantalla de acceso y la vista general una vez metido el usuario y contraseña.

Ilustración 7: Acceso Dashboard Openstack.

Ilustración 8: Vista General de Dashboard Openstack

En esta pantalla se puede, de un solo vistazo, ver el estado general de los recursos libres y usados. A continuación de esos datos, se muestra un resumen de las instancias levantadas.

4.2.1 ¿Cómo crear una máquina virtual?

La primera tarea que se nos presentó en el proyecto fue la provisión de máquinas virtuales para comenzar a instalar en ellas la base de datos y comenzar a hacer pruebas en las mismas. El proceso no es complicado:

1. Ir a la pestaña del lado izquierdo "Instancias" y pulsar "+ Lanzar Instancia"

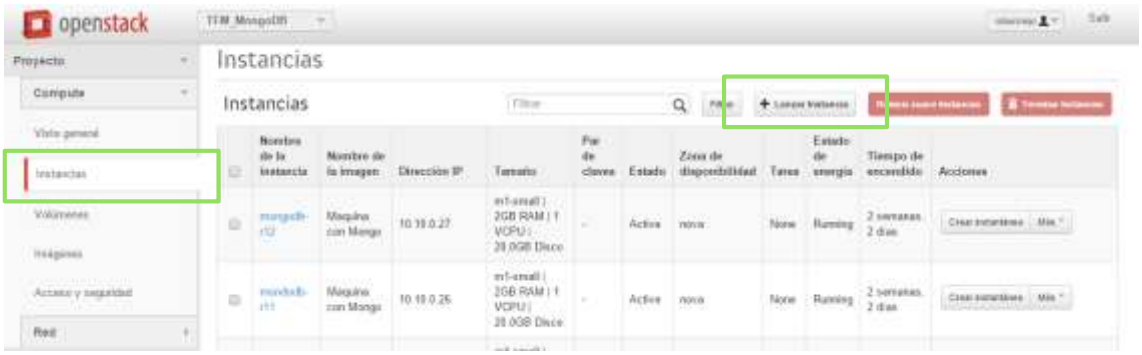


Ilustración 9: Instancia en Dashboard Openstack.

2. Los datos que nos pide, según las pestañas, son:
   i. Zona en la que vamos a crear la instancia. Si tenemos conectado Openstack con otros servicios, es decir otras nubes, nos saldrán distintas zonas de disponibilidad donde queremos crear la instancia.
   ii. Nombre de la instancia.
   iii. Sabor o Flavor que viene determinado por recursos de vCPU, RAM, espacio en disco según la opción que elijamos.
   iv. Opción de arrancar la instancia desde una imagen que tengamos guardada o desde un snapshot que hayamos creado de otras máquinas (como ha sido nuestro caso, al lanzar todas las máquinas ya con la base de datos montada de la primera máquina que se montó, y el snapshot que se hizo).
   v. En la pestaña de Acceso y Seguridad podremos elegir un par de claves que hayamos creado para el acceso a dichas instancias.
   vi. En redes podremos asignar las redes privadas y públicas que tengamos.
   vii. En post-creación podremos insertar un script que queramos lanzar una vez creada la máquina.

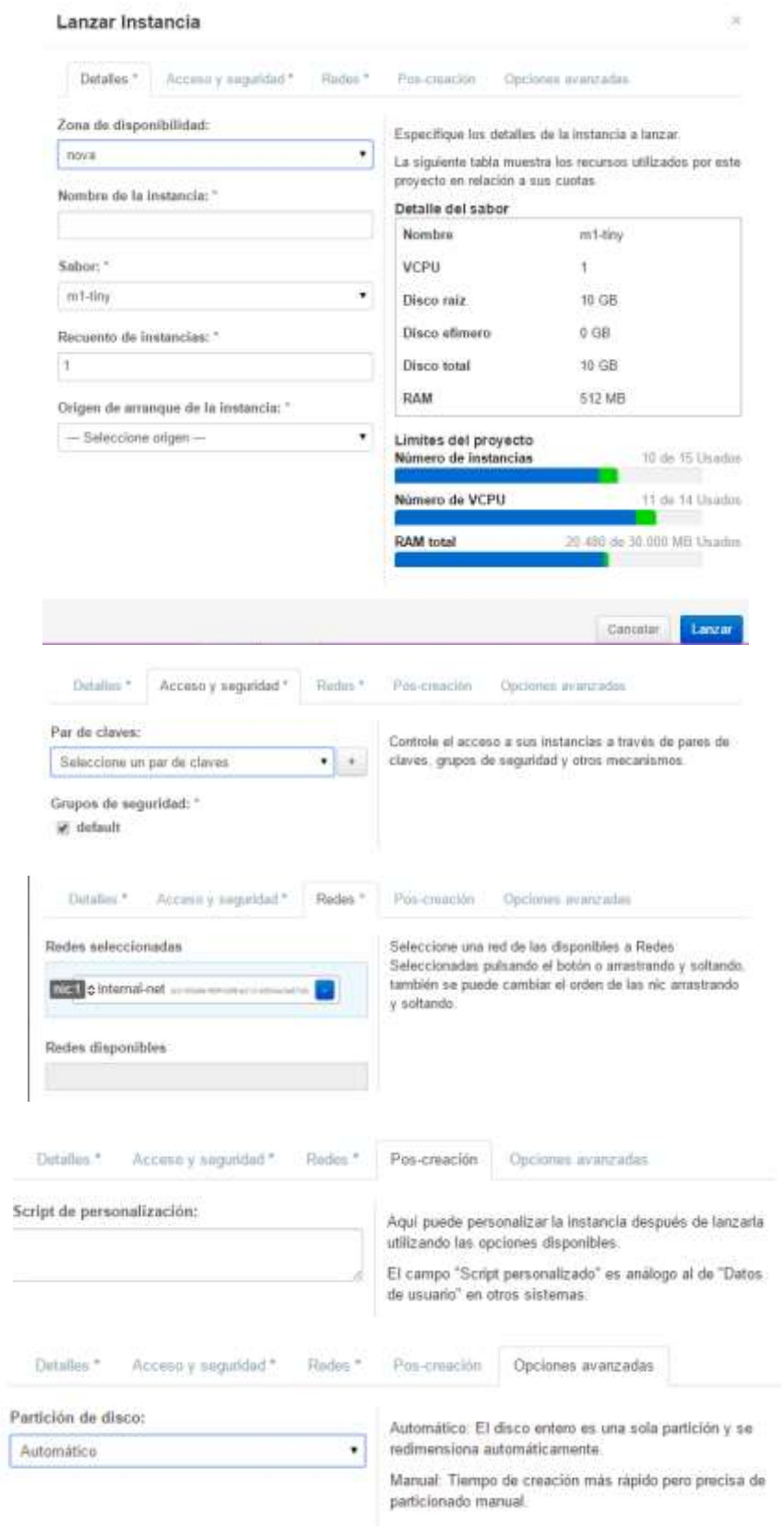


Ilustración 10: Lanzar máquina

Se ha utilizado para todas las máquinas el mismo sabor, small, con las características: 1vCpu, 20Gb de disco y 2Gb de RAM. La red elegida ha sido la única disponible, interna. En el script de *post-creación* se inserta un código para el acceso por contraseña y en pasos posteriores se hace uso de un par de claves para acceso a las máquinas virtuales remotamente.

En la pestaña de "Volúmenes" se puede visualizar el volumen que tenemos asignado:

En "Imágenes", como hemos comentado, se tienen todos los sistemas operativos guardados, o snapshot que hagamos a cualquiera de las máquinas, ya sea como en nuestro caso para ahorrar trabajo en la instalación de servicios, o bien por seguridad y tener una copia para poder restaurar.

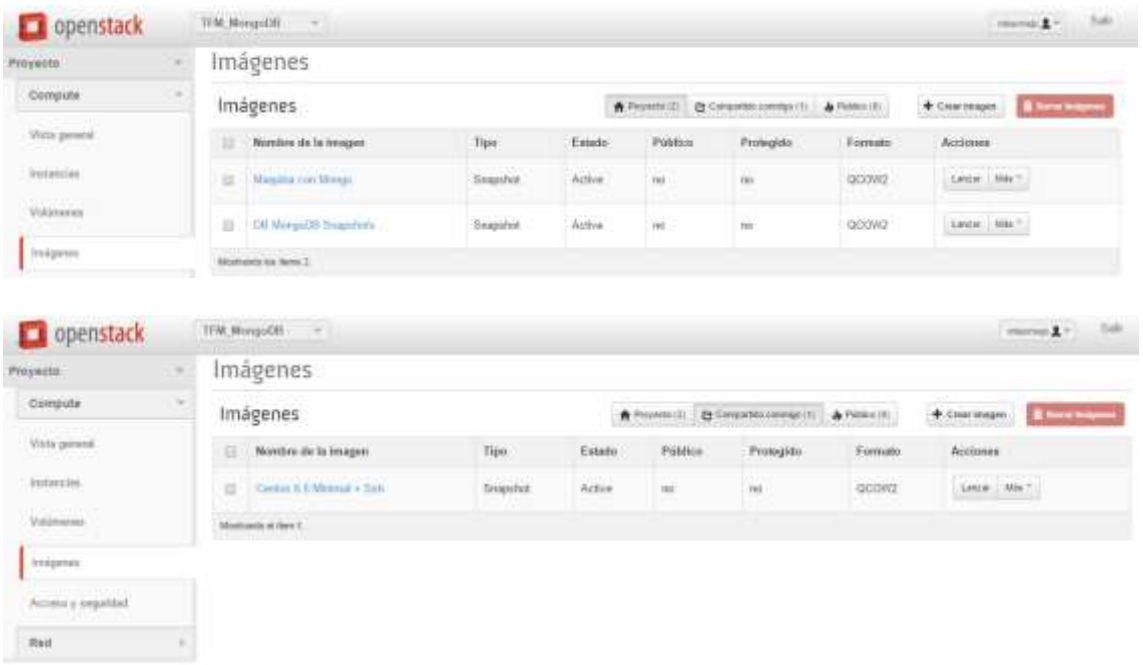


Ilustración 11: Volumen en Dashboard Openstack

En “Acceso y seguridad” nos encontramos con grupos de seguridad que hayamos creado. Aquí hay muchas reglas de seguridad que podríamos configurar, por ejemplo:

Por dirección entrante, saliente; por puerto, por CIDR (enrutamiento entre dominios sin clases) y si estamos en remoto por CIDR o por grupo de seguridad.

Ilustración 12: Reglas que configurar en Dashboard

También nos encontramos el par de clave creado, las IPs flotantes que nos hayan asignado, y las API para acceso a los distintos proyectos:

Ilustración 13: APIs en Dashboard.

Y por último en “Red” nos aparecerá una topología con las redes públicas, internas y máquinas conectadas.

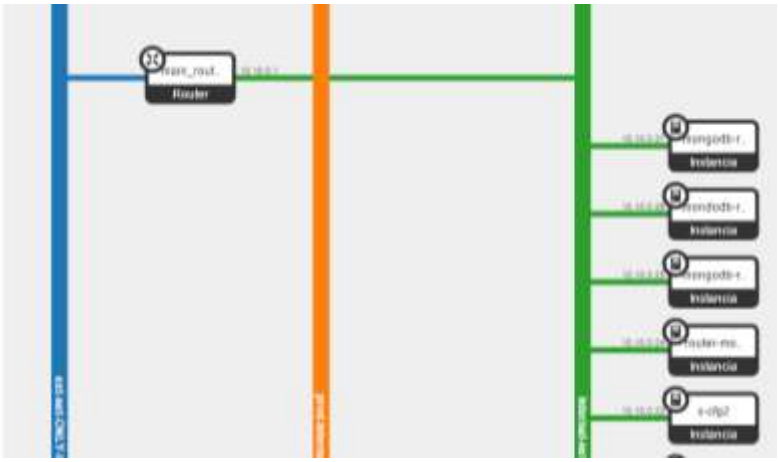


Ilustración 14: Pestaña "Red" en Dashboard Openstack.

Una vez creada la máquina, aparecerá en la "Vista general" y en "Instancias", pudiendo acceder a ella y ver el detalle de la instancia, los logs generado y acceder a la consola de la máquina.

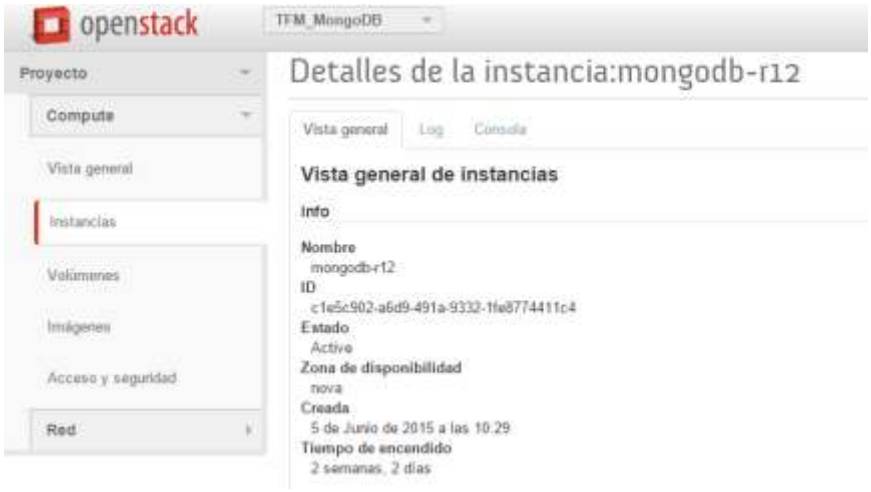


Ilustración 15: Detalle de la instancia Creada en Dashboard de Openstack.

4.3 INTRODUCCIÓN MONGODB

[6] **MongoDB** es una base de datos (DB) multiplataforma orientada a documentos. Esto quiere decir que, en lugar de guardar los datos en registros, guarda los datos en documentos y éstos a su vez en colecciones. Estos documentos son almacenados en BSON, que es una representación binaria de JSON. Esto brinda diversos beneficios tales como un mejor mapeo hacia objetos, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil, rápida y flexible y que tenga una fácil gestión y codificación [7]. Una de las diferencias más importantes con respecto a las bases de datos relacionales es que no es necesario seguir un esquema. Los documentos de una misma colección pueden tener esquemas diferentes. Imaginemos que tenemos una colección a la que llamamos "*Viajes*". Un documento podría almacenarse de la siguiente manera:

```
{
  Lugar: "India",
  Fecha: "22/1/2015",
  Ciudades: ["Bombay","Delhi","Bangalore",”Calcuta”,”Jaipur”],
  Viajeros: [
   {  Nombre:"María", Edad:24  },
   {  Nombre:"Javier", Edad:27  }
  ]
}
```

El documento anterior es un clásico documento JSON. Tiene strings, arrays, subdocumentos y números. En la misma colección podríamos guardar un documento como este:

```
{
  Lugar: "Tetuán",
  Excursion: "Cabo Negro",
  Acompañantes: 4
}
```

Este documento no sigue el mismo esquema que el primero. Tiene menos campos, algún campo nuevo que no existe en el documento anterior e incluso un campo de distinto tipo.

MongoDB viene de serie con una consola desde la que podemos ejecutar los distintos comandos. Esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando ese lenguaje. Además de las funciones de MongoDB, podemos

utilizar muchas de las funciones propias de JavaScript. En la consola también podemos definir variables, funciones o utilizar bucles.

4.3.1 Instalación de MongoDB

Para la instalación de MongoDB disponemos de una MV *CentOS 6.5* creada desde el Dashboard de Openstack con las siguientes características:

Nombre de la instancia	VCPU	Disco	RAM
DB MongoDB	1	20	2GB

Accedemos a la instancia CentOS. En nuestro caso hemos creado y asignado a dicha instancia una IP flotante para poder acceder a ella por SSH, y desde ésta accederemos a las demás por el mismo protocolo a medida que se vayan creando.
Creamos un repositorio en `/etc/yum.repos.d/mongodb.repo` con la siguiente información, para que se instale el paquete correspondiente a la versión 2.6.4 en nuestro caso:

```
[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enable=1
```

Este paquete instala:

`mongodb-org-mongos-2.6.4-1.x86_64`

Contiene el demonio mongos. Este proceso será necesario en el caso de que en nuestra instalación tengamos un Sharded Cluster. Es el responsable de enrutar las peticiones del cliente a la máquina que contiene los datos solicitados y, también, de devolverlos. Es capaz, incluso, de realizar algunas operaciones sobre ellos antes de hacerlos llegar al cliente.

`mongodb-org-server-2.6.4-1.x86_64`

Contiene el demonio mongod, la configuración asociada y scripts de inicialización. Es el proceso que se encarga del funcionamiento de las tareas propias de la base de datos. Estará presente en cada máquina en la que corra la base de datos, ya sea en una máquina aislada o como parte de una Replica Set.

`mongodb-org-shell-2.6.4-1.x86_64`

Paquete que contiene la mongo shell con la que nos conectamos a la base de datos, para tareas administrativas fundamentalmente.

`mongodb-org-tools-2.6.4-1.x86_64`

Conjunto de utilidades que permiten: exportar pequeñas cantidades de datos a fichero (mongoexport), importar datos desde fichero (mongoimport), hacer backups (mongodump), restaurar backups (mongorestore), comprobar el estado de una instancia (mongod o mongos), etc.

¿Cuáles son los directorios de MongoDB?

- Fuentes: /usr/bin
- Logs: /var/log/mongodb (donde creará un archivo mongod.log)
- Datos: /var/lib/mongo

La configuración de parámetros la realizaremos en el archivo: `/etc/mongod.config`. Podemos ver algunos de esos parámetros configurables a continuación.

```
logpath=/var/log/mongodb/mongod.log #guardará todos los log creados en este fichero
logappend=true #reescribe el fichero log
fork=true #permite que el proceso mongod corra como un demonio en background
port=27017  #puerto por defecto que se debe cambiar según la configuración
dbpath=/var/lib/mongo #path donde se guardan archivos relacionados con las DBs
pidfilepath=/var/run/mongodb/mongod.pid  #guarda el PID del proceso daemon
#bind_ip=0.0.0.0  #es la IP a la que mongodb se une para conectarlo a aplicaciones.
```

Una vez que se instala y ejecutamos el proceso `mongod`, éste estará corriendo en la instancia local de MongoDB, comprobándolo de la siguiente manera:

```
[root@mongodb ~]# mongod –f /etc/mongod.conf
ps aux | grep mongo
mongod    1188  0.2  6.4 1155388 123228 ?      Sl   Apr15  56:35
/usr/bin/mongod -f /etc/mongod.conf
```

Y podremos acceder al Shell únicamente insertando por consola `mongo`

```
[root@mongodb ~]# mongo
MongoDB shell version: 2.6.4
connecting to: test
>
```

*Por defecto, siempre MongoDB nos conectará a la db de prueba “test”. Con el comando `use` podremos cambiar a cualquiera de las demás (tanto si están creadas como si no, porque al insertar un documento después del `use` se queda creada).

4.3.2 Inserción de datos

Para insertar documentos, tenemos varias formas de hacerlo según dónde y cómo estén los datos.

- Inserción manual desde el Shell con el formato:
  `db.nombre_coleccion.insert(objeto). Ejemplo:`

```
db.prueba insert
{
    “name” : “Maria”,
    “hobbies” : [ “viajar”, “leer”, “salir” ],
    “amigos” : [
  nombre     Javier   “ocupacion” : “empresario”
                      nombre     Nicolas   “ocupación” : “estudiante”

}
```

- Importar desde fichero JSON en la consola:
  `mongoimport --collection colección --file collection.json`

- Con JAVASCRIPTS.

```
Load(“miarchivojs.js”)
```

*Como *mongo* es un Shell interactivo de JavaScript para MongoDB, para evaluar código JavaScript desde líneas de comando (únicamente 1 comando) se utiliza:

- `mongo test --eval “printjson(db.getCollectionPrueba())”`
  Este comando retorna el dato Javascript correspondiente, pero no mostrará cualquier salida. En este caso, `Printjson()` puede utilizarse para mostrar resultados.
- `Mongo localhost:27017/db --username='username' --password='pwd' mificherojs.js`
  Para ejecutar un fichero con código JavaScript.

A continuación se muestran algunos ejemplos de los comandos equivalentes Shell-JAVASCRIP:

Shell Helpers	JavaScript Equivalents
show dbs, show databases	db.adminCommand('listDatabases')
use <db>	db = db.getSiblingDB('<db>')
show collections	db.getCollectionNames()
show users	db.getUsers()
show roles	db.getRoles({showBuiltinRoles: true})
show log <logname>	db.adminCommand({ 'getLog' : '<logname>' })
show logs	db.adminCommand({ 'getLog' : '*' })

it	cursor = db.collection.find()
	if (cursor.hasNext()){
	cursor.next();
	}

Ilustración 16: Correspondencia entre comando Shell y JavaScript

***NOTA**: En el **Anexo A.I** se muestra la inserción de los datos en MongoDB.*

4.3.3 Almacenamiento de Datos en MongoDB

En MongoDB, como ya se ha mencionado, no existe un esquema estándar para trabajar con los datos, pero eso no significa que vayamos a tener una cantidad de datos difíciles de relacionar. De hecho, la mayor parte del tiempo se trabaja con documentos estructurados, solo que no siguen el mismo esquema todos ellos, sino que cada uno puede tener el suyo propio.

La unidad básica de almacenamiento son los documentos (estos documentos son los que sustituyen a lo que anteriormente conocíamos como fila en los sistemas SQL). Su estructura es la siguiente:

Ilustración 17: Documento MongoDB.

El conjunto de documentos se agrupa en colecciones (en comparación con los sistemas SQL, una colección vendría a ser el equivalente de las tablas):

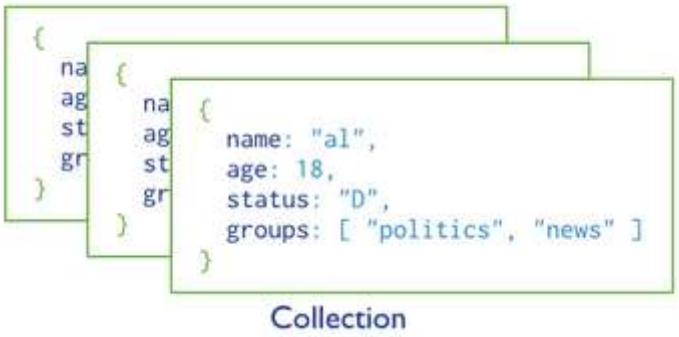


Ilustración 18: Colección MongoDB.

MongoDB corresponde a una Bases de datos documental donde se utiliza una clave única para cada registro.

Este tipo de implementación permite, además de realizar búsquedas por clave–valor, realizar consultas más avanzadas sobre el contenido del documento y ser más versátiles.

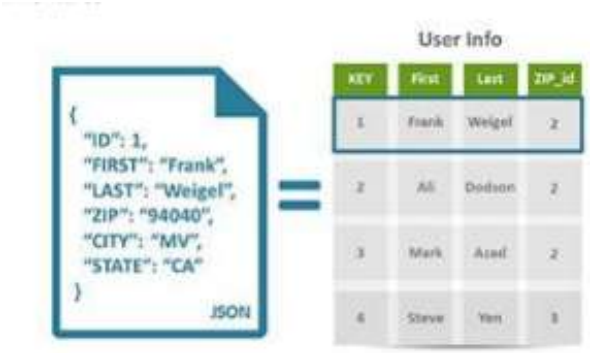


Ilustración 19: Clave-Valor MongoDB

En el ejemplo podemos ver que cada clave posee un valor y dichos valores tienen diferentes tipos de datos: strings, entero, array (aunque las claves siempre son de tipo string). Los pares clave–valor se separan entre ellos por comas y cada clave debe ser única, siendo esta sensible al uso de mayúsculas y minúsculas. Además, MongoDB asigna a cada documento un número de identificación (_id) único dentro de su colección. La gran ventaja de las colecciones es su flexibilidad, ya que los documentos que contienen no tienen por qué poseer el mismo formato. Las colecciones se agruparán en bases de datos, pudiendo albergar Mongo varias bases de datos en una misma instancia.

*Por defecto, el tamaño de los documentos BSON es de 16MB, si se desea superar esta cantidad es necesaria la utilización de GridFS.

4.3.3.1 GridFS

GridFS es una especificación para almacenar y recuperación de archivos BSON que excedan el límite de 16Mb. GridFs divide los ficheros en partes y los almacena en documentos separados con un tamaño por defecto de 255K. La manera en que se almacenan las partes es en dos colecciones: una utilizada para almacenar los trozos de ficheros (`fs.chunks`) y otra utilizada para almacenar los ficheros de metadatos (`fs.files`).

Cuando se consulte un archivo almacenado con GrisFS, el cliente ensamblará los trozos según sea necesario. Es posible realizar búsquedas por rango en los archivos almacenados a través GridFS. También puede acceder a la información de las secciones arbitrarias de archivos.

Se puede elegir una extensión diferente y crear múltiples extensiones en una misma DB. Cada documento en la colección `fs.chunks` representa una parte diferente, identificada por su *Ob_ID*.

Cuando se ejecuta MongoDB utilizando este mecanismo de *Journaling*, tanto el almacenamiento como las aplicaciones escriben las operaciones en memoria en el disco *journal* antes de que los cambios estén presentes en disco sobre los archivos de datos, garantizando la durabilidad en la operación de escritura y resistencia ante *crash*. Antes de aplicar un cambio en los archivos de datos, MongoDB escribe la operación de cambio en el *journal*. Si MongoDB debe terminar o detecta algún error, puede volver a solicitar la operación de escritura y mantener un estado coherente.
Sin *Journaling*, si `mongod` cierra inesperadamente, debe asumir que sus datos están en un estado incoherente y debe funcionar con la reparación o hacer un `resync` de un miembro del conjunto de réplicas.

4.3.3.2 Mecanismo Journalist

MongoDB crea un subdirectorio para albergar *archivos journal* que contienen registro para rehacer escrituras anticipadas y archivos con el número de la última secuencia. Un correcto cierre elimina todos los archivos.

Los archivos tienen extensión `j._` con un máximo de 1Gb por archivo, creándose nuevos archivos llegados a esa capacidad. Estos archivos se eliminan cuando MongoDB aplica todas las operaciones de escritura a los archivos de datos de la DB.

*Para limitar el tamaño de los *archivos journal* utilizar `storage.smallFiles` pudiendo elegir archivos de 128Mb.

Vistas de almacenamiento de la configuración de los datos:

- *Vista de almacenamiento compartida*: Cuando se ejecuta la instancia con *journal*, MongoDB pregunta al sistema operativo si mapea los datos existentes en el archivo de datos en disco a la memoria virtual del compartido. Estas preguntas se lanzan a intervalos predeterminados de 60 segundos, que puede modificarse especialmente si el sistema tiene poca memoria libre.
- *Vista de almacenamiento privada*: para datos de operaciones de lectura. Es el primer sitio donde MongoDB aplica las nuevas operaciones de escritura (en RAM) para después copiar los cambios al almacenamiento compartido donde estarán disponibles para cargarse en los archivos de datos.

4.3.4 Métricas de almacenamiento

MongoDB proporciona gran variedad de indicadores relacionados con el tamaño de su DB. Comprendiendo cada uno de ellos será mucho más fácil comprender en qué estado se encuentra para monitorizar después estos datos.

ARCHIVOS DE DATOS Y EXTENSIONES

Cada instancia genera en el `/dbpath`:

- Namespace .ns. (120 bytes de `<BaseDeDatos>.<colección>`)
- Archivo *journal* o de cambio.
- Archivo de datos.

Para dicho apartado nos centraremos en el archivo de datos, el cual almacena documentos, índices y metadatos llamados extensiones. Las extensiones son contenedores lógicos para almacenar documentos e índices. La relación entre ambos se ilustra en:

Ilustración 20: Relación entre documentos e índices MongoDB.

- ✓ Los datos y los índices se encuentran en cada uno de sus propios conjuntos de extensiones. Nunca contenidos dentro de la misma extensión.
- ✓ Los datos e índices de una colección generalmente abarcan varias extensiones.
- ✓ Cuando se necesita una nueva extensión, MongoDB intentará utilizar el espacio disponible dentro de los archivos de datos actuales, si no creará nuevos archivos de datos.

dataSize

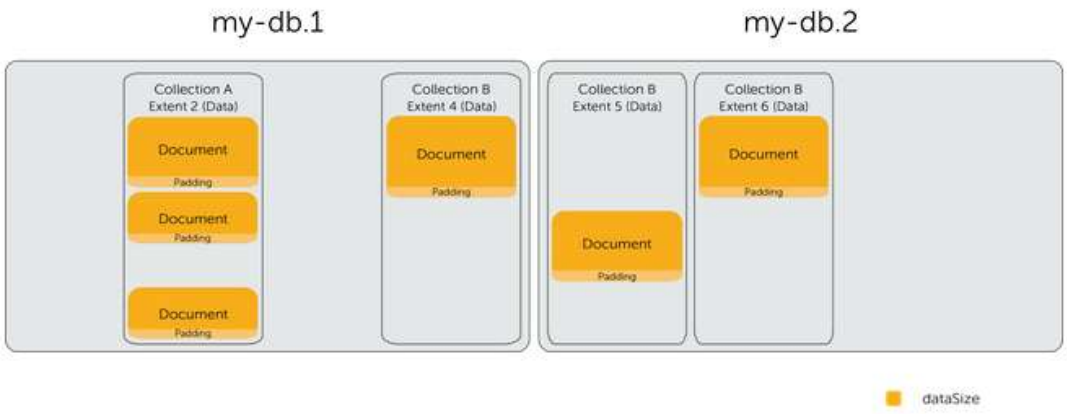


Ilustración 21: dataSize MongoDB

storageSize

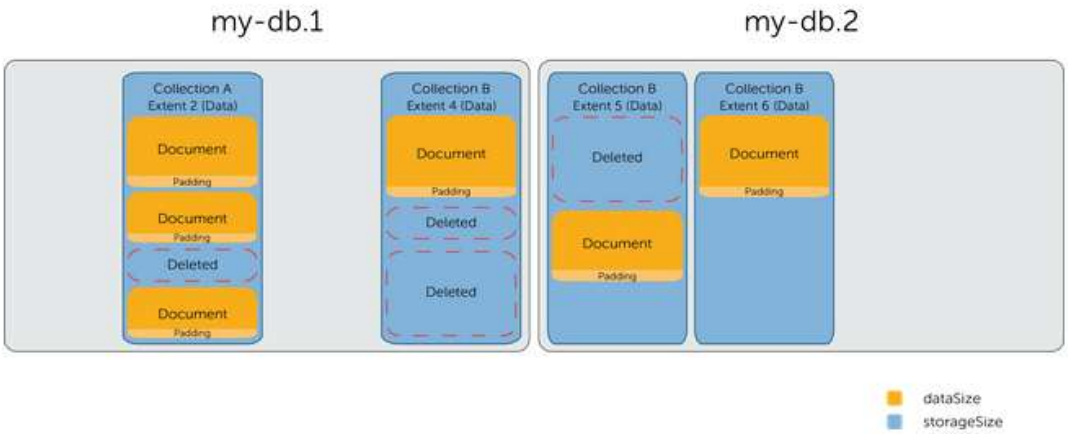


Ilustración 22: storageSize MongoDB

fileSize

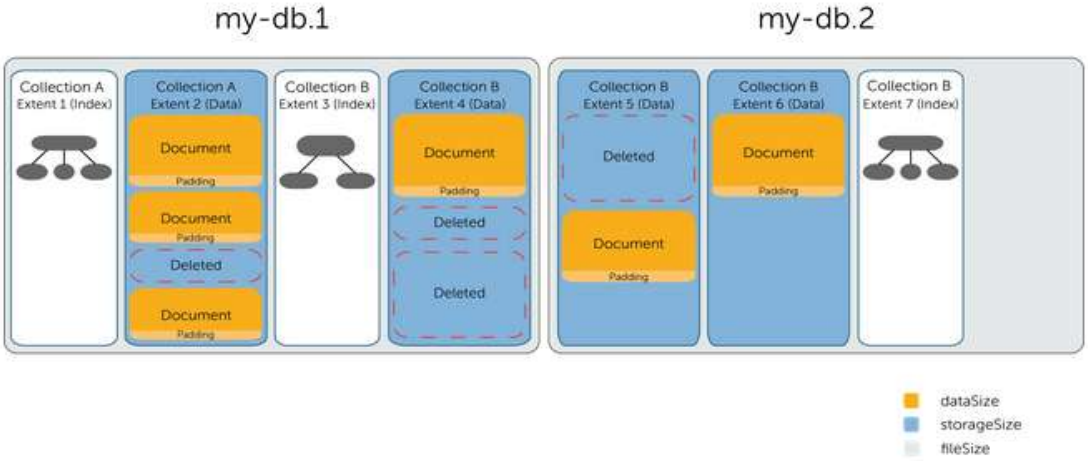


Ilustración 23: fileSize MongoDB

dataSize, storageSize, fileSize

dataSize son los datos contenidos en la base de datos. Disminuye cuando se eliminan documentos, pero no cuando estos se reducen, debido a que el espacio utilizado por el documento original ya ha sido asignado (a ese documento en particular) y no pueden ser utilizados por otros documentos. Alternativamente, si un usuario actualiza un documento con más datos, *dataSize* seguirá siendo el mismo, siempre y cuando el nuevo documento encaje dentro de su espacio de pre-asignado (padding incluido).

storageSize es igual al tamaño (en bytes) de todas las extensiones de datos en la base de datos. Este número es mayor que dataSize, ya que incluye el espacio todavía-no utilizado (en extensiones de datos) y el espacio vacante por documentos eliminados o movidos dentro de extensiones. El valor storageSize no disminuye al quitar o reducir documentos.

fileSize es igual al tamaño (en bytes) de todas las extensiones de datos, extensiones de índice y espacio todavía no utilizado (en archivos de datos) en la base de datos. Se representa el espacio de almacenamiento en el disco. Es mayor que *storageSize* porque incluye extensiones de índice y espacio todavía no utilizado en los archivos de datos. *fileSize* no disminuye quitando colecciones, documentos o índices. Sin embargo, disminuirá si elimina una base de datos o ejecutar `db.repairDatabase()`, una operación que desfragmenta y libera espacio.

4.3.5 Monitorización

El monitoreo es la tarea crítica de toda la administración de DBs. Permite evaluar el estado de la base de datos y no provocar crisis en los despliegues de infraestructuras cuando no sea necesario.

Hay tres métodos para la recopilación de datos sobre el estado de una instancia de MongoDB funcionando:

- Utilidades distribuidas con MongoDB que proporciona información en tiempo real de las actividades de base de datos.
- Comandos de base de datos que devuelven estadísticas sobre el estado de la base de datos.
- Y una solución de Empresa Avanzada de MongoDB que es el MongoDB Servicio de Gestión (MMS) , un servicio alojado que proporciona supervisión para recopilar datos de funcionamiento en despliegues MongoDB, así como "Ops Manager" y la visualización y alertas basados en estos datos. [8]

Comencemos por ver algunas utilidades que nos van a ayudar en el progreso del proyecto:

`mongostat`

Captura y muestra la cantidad de operaciones en la base de datos por segundo (en configuración general). Esta cantidad nos da la carga distribuida en el servidor por tipo de operaciones, e informa de la capacidad.

Opciones (relevantes):

`<sleeptime>` | Cantidad de tiempo, en segundos, que mongostat espera entre llamadas. Por defecto cada segundo

`--rowcount <number>, -n` | Controla el nº de filas. En su ausencia mongostat retorna infinitas filas.

Salidas:

```
insert  query update delete getmore command flushes mapped  vsize    res faults

locked db idx miss %     qr|qw   ar|aw  netIn netOut  conn       time
```

Ilustración 24: Salidas de `mongostat`

`mongotop`

Reporta la actividad lectura/escritura actual de la instancia y nos da una estadística por colección.

Opciones (relevantes):

La cantidad de salidas por segundo.

Salidas:

```
                              ns       total        read       write
2015-05-11T19:09:55
          admin.system.indexes          0ms         0ms         0ms
       admin.system.namespaces          0ms         0ms         0ms
          admin.system.profile          0ms         0ms         0ms
            admin.system.roles          0ms         0ms         0ms
            admin.system.users          0ms         0ms         0ms
          admin.system.version          0ms         0ms         0ms
```

Ilustración 25: salida de `mongotop`

Seguimos con Comandos de Base de Datos (Diagnostic Command):

`dbstat()`

Devuelve una estadística del estado de uso de la base de datos. Como opciones se puede especificar el valor de escala que mostrará los resultados, por defecto en bytes (opciones).

Se muestra un ejemplo conectándonos a una DB desde consola y utilizando el comando `db.stats()` y para una colección `db.pruebai.stats()`.

*Importante: hay que conectarse a la base de datos (comando `use`), si no, no reconoce la colección. La salida, como no hemos especificado nada, nos la da en Bytes.

```
[root@mongodb ~]# mongo maria --
eval
"printjson(db.stats())"
MongoDB shell version: 2.6.4
connecting to: maria
{
        "db" : "maria",
        "collections" : 4,
        "objects" : 20008,

"avgObjSize":111.9808076769,
        "dataSize" : 2240512,
        "storageSize" : 2818048,
        "numExtents" : 8,
        "indexes" : 2,
        "indexSize" : 572320,
        "fileSize" : 67108864,
        "nsSizeMB" : 16,
        "dataFileVersion" : {
        "major" : 4,"minor" : 5
        },
```

```
[root@mongodb ~]# mongo maria --
eval
"printjson(db.pruebai.stats())"
MongoDB shell version: 2.6.4
connecting to: maria
{
        "ns" : "maria.pruebai",
        "count" : 1,
        "size" : 48,
        "avgObjSize" : 48,
        "storageSize" : 8192,
        "numExtents" : 1,
        "nindexes" : 1,
        "lastExtentSize" : 8192,
        "paddingFactor" : 1,
        "systemFlags" : 1,
        "userFlags" : 1,
        "totalIndexSize" : 8176,
        "indexSizes" : {
                "_id_" : 8176
        },
```

Ilustración 26: Salida comando `db.stats()`

"**db**" : nombre de la DB.

"**collections**" : número de colecciones en la DB.

"**objects**" : Documentos de todas las colecciones.

"**avgObjSize**" : tamaño medio de cada documento. Datasize/object.

"**dataSize**" : tamaño total de los datos contenidos en la base de datos que incluye "factor padding*". La escala afecta a este valor.

"**storageSize**" : cantidad de espacio asignado a colecciones para almacenar documentos. La escala afecta al valor.

"**numExtents**" : nº extensiones en la DB de todas las colecciones.

"**indexes**" : nº de "indexes".

"**indexSize**" : tamaño total de esos "indexes" creados.

"fileSize" : tamaño de los archivos de datos que contiene la DB. Incluye el españo preasignado y el "padding factor". Sólo refleja el tamaño de los archivos para la DB y no el archivo "namespace". Solo presente cuando se usa el almacenamiento *mmapv1.*

"**nsSizeMB**" : tamaño total de los archivos "namespace" (terminación .ns). No se puede cambiar el tamañao de éstos una vez creada la DB, pero se puede cambiar el tamaño por defecto con la opción snSize. Solo presente cuando se usa el almacenamiento *mmapv1.*

"**dataFileVersion**" : formato on-disk (mayor y menor). Solo presente cuando se usa el almacenamiento *mmapv1.*

"**extentFreeList**" : num: nº extensiones en la freelist y size: nº total de extensiones. Solo presente cuando se usa el almacenamiento *mmapv1.*

Para el `collstat,` que es el mismo comando, solo que centrado en una colección y no en una base de datos:

"**userflags" :** indica la configuración de usuario en la colección. Solo para *mmapv1*

0 corresponds to **usePowerOf2Sizes** flag set to false and **noPadding** flag set to false.
*1 corresponds to **usePowerOf2Sizes** flag set to true and **noPadding** flag set to false.
2 corresponds to **usePowerOf2Sizes** flag set to false and **noPadding** flag set to true.
3 corresponds to **usePowerOf2Sizes** flag set to true and **noPadding** flag set to true.

**Padding Factor* es una constante que se utiliza para determinar qué espacio adicional necesita el documento contenedor en el disco. PaddingFactor=1 significa que asignará el espacio necesario, =2 significa que asignará el doble. Mongo utiliza diferentes estrategias de asignación debido a que los documentos en MongoDB pueden crecer después de la inserción y todos los registros son contiguos. Dicha estrategia puede reducir la necesidad de reubicación de los datos, que es menos eficiente que una actualización del mismo en el lugar asignado, llevando a una fragmentación del almacenamiento.

**Power of 2 Sized Allocations* es la estrategia para *mmapv1* (en versión 3.0. se añade una nueva ingeniería de almacenamiento llamada *wiredTiger*). Cada registro tiene un tamaño múltiplo de 2 (32, 64, 128, 256, 512….2MB). Para mayores de 2Mb se redondea al múltiplo más cercano. Ventajas:

- Reduce la fragmentación utilizando espacio liberado. Aumenta la posibilidad de que una inserción encaje,
- Reduce movimientos por el relleno incluido.

Hablamos por último de la solución de Empresa Avanzada de MongoDB: MongoDB Servicio de Gestión (MMS). Para su instalación el proceso ha sido instalar unos agentes proporcionando la IP a la cual conectarse, y el mismo programa desde la página web lo configura.

La interfaz es la siguiente:

Ilustración 27: Interfaz de MMS

Nosotros utilizaremos solo la parte de monitorización, pero es una herramienta muy potente con la cual se pueden hacer tareas de administración creando nuevas instancias, configuración de sharding y replication, etc.

Si pinchamos en "View Metrics" nos salen todas las gráficas que podemos gestionar. Podemos visualizar gráficas del Estado General, o el estado de una DB concreta:

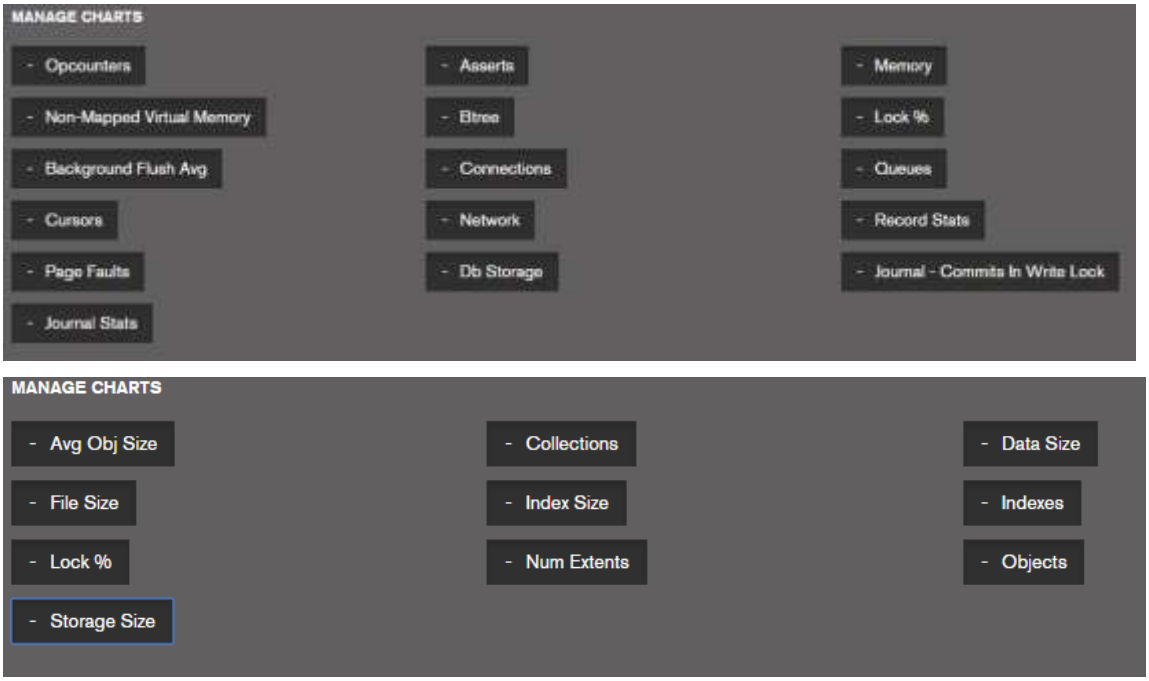


Ilustración 28: Gráficas que pueden mostrarse en MMS

> ***NOTA**: En el **Anexo A.II** se realizan monitoreos con las distintas herramientas y se ofrecen unos resultados de comparación.*

4.4 REPLICA SET, MONGODB

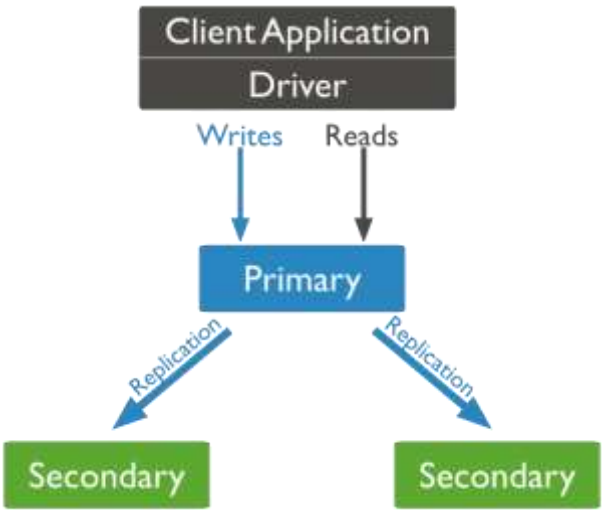


Ilustración 29: Replica Set. Arquitectura

[9] Un *Replica-Set - RS,* es un grupo de instancias `mongod` que albergan el mismo conjunto de datos. Estas instancias podrán ser:

- Primarios: reciben todas las operaciones de escritura.
- Secundarios: copian operaciones del Primario de modo que todos tienen el mismo conjunto de datos.

Únicamente puede haber un Principal por Replica-Set.

Para la comunicación en el conjunto de Replica-Set, todos los miembros se envían heardbeats / pings constantemente.

Para llevar a cabo la replicación, los Primarios graban todos los cambios de sus datos en un archivo denominado `oplog`, que se explica más adelante, que será el que repliquen los secundarios para aplicar los cambios en cuanto a operaciones realizadas.

4.4.1 Introducción

Diferencia entreReplica Set y Maestro/Esclavo

El replica set es una *superconfiguración* funcional del Maestro/Esclavo, nueva y más robusto.

Primario y Maestro son nodos que aceptan operaciones de escritura. La replicación en MongoDB es "*single-master*": solo un nodo puede aceptar escrituras al mismo tiempo.

En un Replica-Set, si el Primario cae o no está accesible, otros miembros pueden elegir a otro miembro para que sea el nuevo primario. Por defecto, todos los clientes leen del primario, pero es configurable para que dichas operaciones se hagan en miembros secundarios.

Secundarios y Esclavos son nodos de sólo lectura que replican del primario. En la replicación se utiliza el `oplog`, desde donde los secundarios aplican las operaciones nuevas sucedidas. Es un proceso asíncrono, que en una red LAN no debe llevar más de unos milisegundos.

Podríamos decir que el Replica set es una forma de Maestro/Esclavo asíncrono, añadiendo tolerancia a fallos o *failover* y recuperación automático.

`Oplog`

El `oplog` (log operaciones) es una colección especial con un tamaño fijo que mantiene un registro de todas las operaciones que modifica los datos almacenados en la DB. Los miembros secundarios que copian y aplican esas operaciones lo hacen mediante un *proceso asíncrono*. Todos los miembros del Replica Set contienen una copia del `oplog` en la colección `local.oplog.rs`. Para ver el estado del oplog (tamaño y tiempo de operaciones) se puede utilizar el comando `rs.printReplicationInfo()`. Para comprobar que la replicación en los secundarios no sufre retraso ejecutar en cualquier miembro, se utiliza `db.getReplicationInfo()`.

Respecto al tamaño de `oplog`, MongoDB crea uno de un tamaño predeterminado (configurable), que dependerá de la arquitectura del sistema operativo. En la mayoría de los casos, el tamaño por defecto del oplog es suficiente. Por defecto:

- 64-bit Linux, Solaris, FreeBSD, y Windows → MongoDB dedica el 5% del espacio libre en disco pero siempre al menos 1Gb y nunca más de 50Gb.
- 64-bit OS X systems → MongoDB dedica 183Mb.
- 32-bit systems → MongoDB dedica sobre 45Mb.

Las cargas de trabajo que podrían requerir un mayor `oplog` son: actualizaciones de varios documentos a la vez, datos insertados igual al número de datos eliminados y actualizaciones de documentos que no se traducen a un incremento de tamaño.

Replicación Asíncrona

La Replicación entre miembros es asíncrona para proporcionar consistencia eventual, es decir, que el sistema propaga gradualmente los cambios, no siendo necesario tener miembros legibles para reflejar las últimas escrituras. Esto lleva a que:

- Las lecturas en un Primario tienen consistencia estricta.
- Las lecturas en los Secundarios tienen consistencia eventual.

**Journal* proporciona la durabilidad de la escritura en instancias simples.

Mongo usa dos formas de sincronización de los datos:

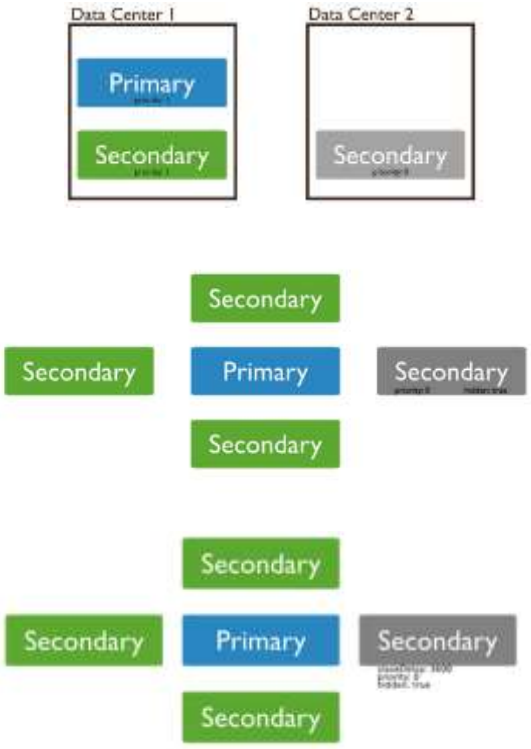
- *Sincronización inicial*: copia todos los datos de uno de los miembros del RS a otro miembro. Esta sincronización se utiliza cuando un miembro no tiene datos, es nuevo o tenía pero faltan antecedentes de la replicación.
- *Replicación*: los miembros del RS replican continuamente después de la sincronización inicial. Este proceso guarda los miembros actualizados con todos los cambios en los datos del RS. En la mayoría de los casos, los Secundarios sincronizan desde el primario pero pueden cambiar si es necesario basándose en los tiempos de ping y estado de los otros miembros. Para ello es necesario que ambos miembros tengan el mismo valor de `buildIndexes`.

4.4.2 Miembros de un RS

Primario. Solo puede haber uno por RS.

Secundarios. Este miembro puede configurarse para un objetivo específico. Esta configuración puede ser:

- ✓ Impedir que se convierta en un Primario en una elección, lo que permite que resida en CPD Secundario o que sea standby en frío: `Priority 0.`
- Miembro del RS Oculto con copia de datos del primario pero invisible a aplicaciones de cliente. Esto permite ejecutar aplicaciones que requieran separación de tráfico. Los miembros ocultos se usarán para tareas como reporte o backup. `mongod` no interatúa con estos miembros.
- Guarda un snapshot "histórico" en ejecución para usarlo en recuperación de ciertos errores como eliminación de DB intencionadamente. Miembros Delayed, contiene copias del conjunto de réplicas pero con un retraso respecto al conjunto; dicho retraso se especifica en `slaveDelay.`



Ilustraciones 30: Configuraciones de Secundario de un Replica - Set.

Árbitro. Miembro que no tiene copia de los datos y no puede ser Primario, pero añade un voto a la elección de un nuevo Primario. Esta configuración favorece las situaciones en las el presupuesto es limitado porque este miembro no requiere HW dedicado, y se podrá ejecutar en un servidor de otro proceso.

El requisito mínimo para establecer un Replica-Set es: un Primario, un Secundario y un Árbitro, aunque en la mayoría de los desarrollos se establece: un primario y dos secundarios.

En la versión que utilizamos para dicho proyecto puede tener hasta miembros. En versiones posteriores (3.0.0) se pueden incluso llegar a 50 miembros, pero solo 7 podrían votar en situaciones de *failover*.

Arquitecturas de desarrollo. Estrategias.

Nº impar de Miembros.

Esto asegura que el RS siempre sea capaz de elegir un primario. Si hay un nº par, añade un árbitro para conseguir ese nº impar.

Tolerancia a fallos

Diferencia entre el número de miembros del RS y el número necesario para la elección de un primario. Esta relación no es directa:

Número de miembros.	Mayoría necesaria para elegir un nuevo Primaria.	Tolerancia a fallos.
3	2	1
4	3	1
5	3	2
6	4	2

Miembros ocultos y retardados para funciones dedicadas, como copias de seguridad o presentación de informes.

Distribución de miembros secundarios, para mejorar la redundancia y disponibilidad en despliegues con alto tráfico de lectura.

Journaling para fallos de energía.

4.4.3 *Failover* y Elección en un RS del primario

La elección es fundamental para el funcionamiento del RS, ya que mientras se está dando dicha elección y asignación de primario, el conjunto no tiene primario y por consiguiente no puede aceptar escrituras, y los demás miembros siguen estando en modo lectura.

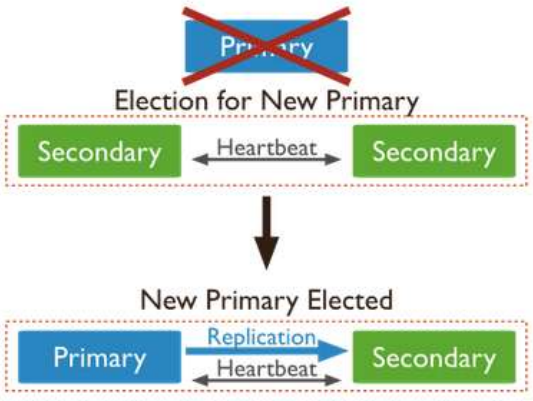


Ilustración 31: Failover en el Replica Set

Los miembros del RS envían ping/heardbeats entre sí cada dos segundos. Si un ping no retorna en 10s, los otros miembros lo marcan como inaccesible.

Para la votación, los miembros elegirán siempre a quien tenga el `Priority` o prioridad más alta.

¿Quién puede o no ser Primario?

- ✓ Quien tenga el `Optime`, fecha y hora de la última operación aplicada al `oplog`, actualizado.
- ✓ Quien tenga Conexión con la mayoría de los miembros del RS.
- ✓ Las particiones de red afectan para la elección. Si el Primario cae, y ninguno de los miembros tiene mayoría, el Replica-Set quedará en modo solo lectura. Para evitar esto, coloca mayoría de instancias en un data-center y minoría en distintos data-center combinados.

Mecánica de elecciones

Eventos para la elección de un primario:

- Iniciación de un nuevo RS
- Secundario pierde comunicación con primario.

  *Los Secundarios con `Priority 0` no convocan elecciones.
- Un primario cae.
- Después de recibir el comando `replSetStepDown` con el que fuerza a un primario a ser secundario convocando elección para primario. El comando hace caer al primario por unos segundos especificados.
- Si un secundario puede optar a ser elegido y tiene `Priority` elevada.

Por defecto, todos los miembros del RS tienen `Priority 1` y tienen las mismas posibilidades de ser primario, además de tener la posibilidad de participar en la elección, concepto también modificable con el comando `votes 0`.

Rollbacks durante la caída de un RS

Un *rollback,* o cancelación, restablece las operaciones de un ex-Primario cuando el miembro se reincorpora al RS después de un failover. Un retroceso es necesario solo si el Primario había aceptado operaciones de escritura que los Secundarios no habían replicado aun. Esto hace que se mantenga la coherencia en la DB.

Cuando acontece un *rollback*, los administradores deben decidir si aplicar o ignorar los datos. Dichos datos se guardan en una carpeta en el directorio `dbPath` en archivos BSON con la siguiente forma: `<Database>. <Collection>. <Timestamp> .bson.`
Se utiliza `bsondump` para leer el contenido y `mongorestore` para aplicar los cambios en el primario convertido a secundario.

Limitaciones: Una instancia mongo no deshace más de 300Mb de datos, si es así debe realizarse manualmente.

Con estas nociones básicas ya podemos comenzar a poner en funcionamiento un RS.

- ***Nota:** En el **Anexo A.III** se muestra paso a paso la configuración de un Replica-Set.*

4.5 SHARDING, MONGODB

[10] Sharding es un método para almacenar datos en varias máquinas dirigidas a desarrollos con mucha carga de datos y operaciones con una capacidad de procesamiento o *throughput* alto en las operaciones.

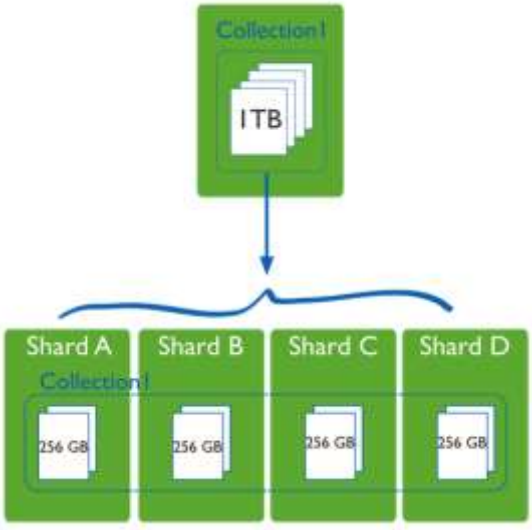


Ilustración 32: Sharding Arquitectura.

El método tradicional para paliar dichas saturaciones es el escalado vertical que añade más CPU, RAM y recursos de almacenamiento para incrementar la capacidad. Pero con la limitación del coste que supone.

El Sharding o escalado horizontal divide los datos y distribuye dichos datos por múltiples servidores o **shards**. Cada shard es una base de datos independiente que colectivamente trabaja como una sola base de datos lógica. Estos shard, que podrán ser nodos standalone o Replica-Sets formarán parte de un Cluster de Shard.

Quien distribuye los datos por cada shard será un enrutador. Dicho enrutador hace de interface única con la parte de aplicaciones de cliente, ya que éstas nunca conectan o comunican directamente con los shards. Este enrutador será una instancia que rastrea los datos que van o están en cada shard cacheando metadatos. Estos metadatos se encontrarán en unos servidores de configuración que serán cada uno de ellos instancias de `mongod`. [11]

Importante: `mongos` no tienen un estado persistente y consume mínimos recursos del sistema.
Ventajas del Sharding:

- ✓ Reduce el número de operaciones que se manejan en cada shard.
- ✓ Reduce la cantidad de datos que cada servidor necesita almacenar.

4.5.1 Introducción

Partición de los datos

La distribución se hace a nivel de colección mediante una clave de shard. Dicha clave puede ser o un campo indexado o un campo compuesto indexado que exista en cada documento de la colección.

MongoDB divide el valor de la clave shard en trozos denominados chunks y distribuye dichas partes por los shards usando: particionado basado en rangos o particionado basado en *hash (visto en apartados posteriores)*.

El tamaño de chunk va desde 1 a 1024 Mb siendo 64Mb el tamaño por defecto en la configuración de `mongos`

Particionado basado en rangos

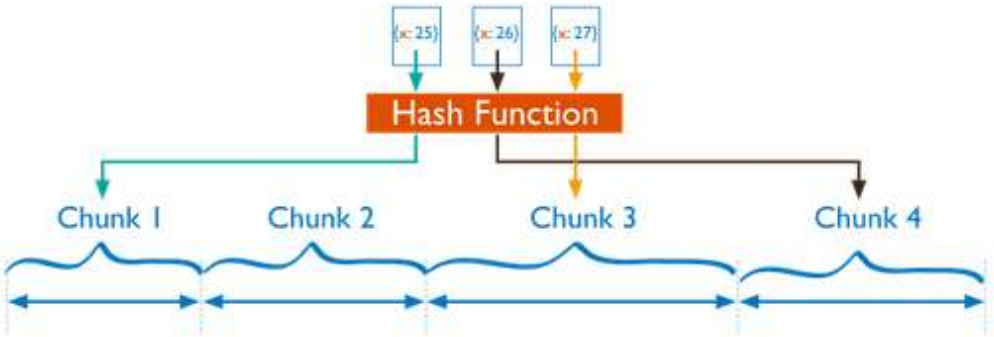


Ilustración 33: Particionado basado en rangos en Sharding

Un chunk es un rango de valores desde un valor mínimo a un valor máximo, nunca superponiéndose. Ventajas:

- Más eficiente por el rango determinado en la clave de shard
- En una rango basado en un rango incremental como tiempo o fechas, si todas las peticiones se hacen para un mismo rango recibiendo la mayoría de las peticiones, el mismo conjunto de shards.

Particionado basado en Hash

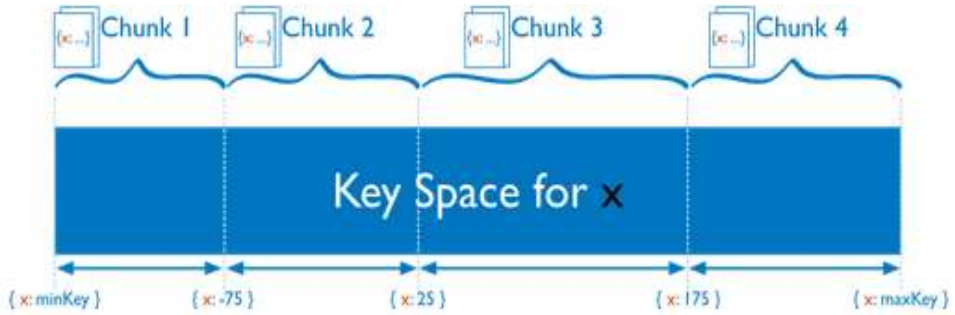


Ilustración 34: Particionado Hash Sharding

Calcula un Hash del valora de un campo, y después usas ese hash para crear chunks. Ventajas:

- Basado en valores aleatorios, con lo que la problemática de consultas en un mismo rango no se daría.
- Las respuestas no van a ser tan eficientes como en el caso del rango ya que es menos probable.

4.5.2 Sharding Key

[12] La clave de shard afecta al rendimiento de operaciones de escritura y peticiones de consulta determinando como particionar los datos en el cluster y el grado de eficacia de `mongos` para dirigir operaciones al cluster, por eso hay que tener en cuenta varias consideraciones a la hora de elegir dicha clave.

Para muchas colecciones puede que no que haya una sola clave, ya que éstas poseen todas las cualidades para ser una buena clave, pero las estrategias siguientes pueden ayudar a construirla:

- Imaginar la mejor clave en la capa de aplicación y guardarla en todos los documentos, preferiblemente en el _id.
- Usar una clave compuesta que use dos o tres valores y que proporcione la combinación adecuada de cardinalidad[1] con operaciones de escritura escalable y consultas aisladas.
- Determinar que el impacto de usar una clave menos ideal es insignificante en nuestro caso de uso, dado:
    - El volumen de escritura.
    - El tamaño del dato.
    - O los patrones de consulta.

[1]Cardinalidad en MongoDB se refiere a disponibilidad del sistema a particionar el dato en chunks.

Una vez aclarada la estrategia a seguir para la elección de la clave, hay una serie de consideraciones a tener en cuenta:

- Crear una clave que sea fácilmente divisible. Esto hace que MongoDB distribuya el contenido entre los shards. Las claves que tengan un número limitado de posibilidades pueden dar como resultado chunks indivisibles que afectan a la cardinalidad.
- Crear claves que tengan alto grado de aleatoriedad, previniendo así que un shard se convierta en cuello de botella, repartiendo las operaciones de escritura por todo el cluster.
- Crear claves que se dirijan a un único shard, haciendo posible que el programa mongos retorne más operaciones de consulta directamente desde una instancia `mongod` específica. La clave debería ser el primer campo usado en las

consultas. Los campos con alto grado de aleatoriedad hacen difícil dirigir consultas a shards específicos.

- ❖ Usar claves compuestas, de forma que no siempre hay una opción obvia. No siempre un campo existente en la colección es la clave óptima.

Como comprobamos, es difícil elegir una clave que proporcione alto grado de aleatoriedad para los datos y que permita que la aplicación dirija las operaciones a shards específicos. Según las cargas de trabajo, será más importante una que otra. Siempre deberá tenerse en cuenta un cierto balance.

Limitaciones:

- ✓ Una clave de shard no puede exceder los 512bytes.
- ✓ El índice puede ser:
  - Índice ascendente.
  - Índice compuesto que comience con la clave de shard y especifique el orden ascendente.
  - Índice hasheado.
- ✓ La claveha de ser invariable. No se puede cambiar la clave después de particionar una colección. Si se desea, hay que volcar todos los datos en un formato externo, eliminar la colección particionada, configurar la nueva clave, (split) y restablecer los datos.

Splitting y Balanceo

Para asegurar un cluster balanceado en cuanto a datos por shard o número y tamaño de chunk por shard, se utilizan dos métodos: splitting y balanceo.

Splitting, es un proceso en background que guarda los chunk que crecen demasiado rápido. Los chunk tienen un tamaño específico (configurable), y MongoDB lo divide a la mitad. Las inserciones y actualizaciones activan el splitting para cambiar la eficiencia del metadato. Para realizar estas divisiones, MongoDB ni migra datos, ni afecta a los shard.

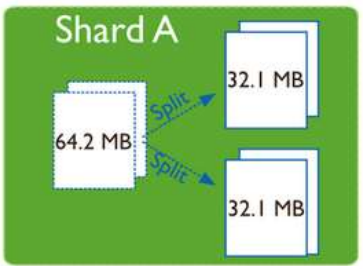


Ilustración 35: Split Sharding

Balanceo, es un proceso en background que maneja las migraciones de chunk ejecutándose desde cualquier router del cluster. Durante la migración, el shard destinatario se envía a todos los documentos actuales del chunk desde el shard original, y cuando finaliza el shard destino aplica todos los cambios hechos durante la migración. Para finalizar, el metadato guardará la nueva localización del chunk en el servidor de configuración.

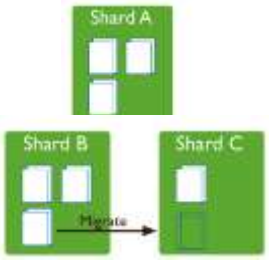


Ilustración 36: Balanceo Sharding

4.5.3 Arquitectura del Cluster de Shards

Mongo soporta Sharding a través de una arquitectura Cluster de Shards, que consiste, para un entorno en producción, en:

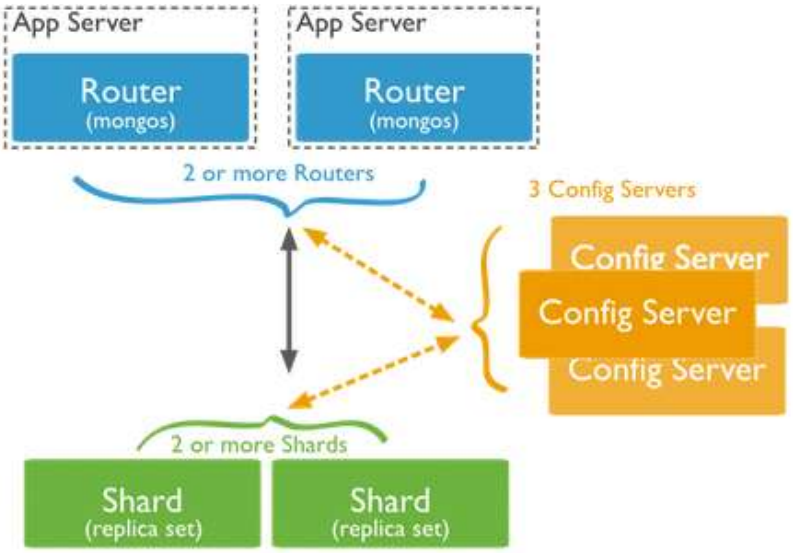


Ilustración 37: Ariquitectura Cluster Sharding

- **Shards:**

  Almacena datos. Generalmente será un Replica-Set para proporcionar redundancia y alta disponibilidad. Cada base de datos tiene un shard primario que mantiene todas las colecciones en las bases de datos. Para cambiar de

shard primero utilizaremos `movePrimery` (que llevará un largo periodo de tiempo hasta completarse). Para comprobar el estado del shard, utilizamos el comando `sh.status()`, que contiene información de la configuración del Sharding e información referente a los chunks existentes en el cluster.

- **Router de peticiones: instancia `mongos`**
  Interface con aplicación cliente y con el shard apropiado según la operación de escritura o consulta requerida. El router procesa y apunta las operaciones al shard adecuado y devuelve al cliente los resultados.
- **Servidores de aplicación: instancias `mongod`**
  Almacena los metadatos del cluster. Contiene un mapping de la configuración de datos de los shards que utilizará el router de peticiones. Los clusters tienen 3 servidores de configuración. MongoDB solo escribe datos al servidor de configuración cuando los metadatos cambian como en siguientes casos: después de una migración chunk y después de una división de chunk (split)

Cuando se escribe en los servidores, se escriben los mismos comandos en todos los servidores y se recogen los resultados. Esta función la hace un coordinador. Una vez que los resultados indican inconsistencia, el balanceador no realizará ninguna migración de chunks, y `mongos` no realizará auto-splits de chunks.
MongoDB lee datos de estos servidores cuando:

- ✓ Se inicializa un `mongos` por primera vez o se reinicia
- ✓ Después de cambios en el cluster o migración de chunks.

Utilizaremos `CNAMEs` para identificar los servidores de configuración.

Requerimientos y Rendimiento de Cluster de Shard

El Sharding es una característica completa y potente pero con una infraestructura compleja con requerimientos y complejidad en su desarrollo. Solo se montará un cluster Sharding si la aplicación lo requiere, como por ejemplo:

- ✓ Si los datos requieren metodología o exceden en la capacidad de una instancia de MongoDB standalone.
- ✓ El tamaño del "*working set*" (datos que se usan a menudo salvados en RAM, SSD u otros medios) excede rápido la capacidad RAM del sistema.
- ✓ Una instancia standalone no soporta la demanda de operaciones de escritura.

Requerimientos:

- ✓ Chunk = 64Mb máx. por defecto.
- ✓ El balanceador no moverá datos por los shard hasta que los chunks.

4.6 NOVA, OPENTACK

Nova es el subsistema más complejo e importante de Openstack. Se compone de muchos procesos, que podemos ver en el siguiente esquema y explicados a continuación. [13]

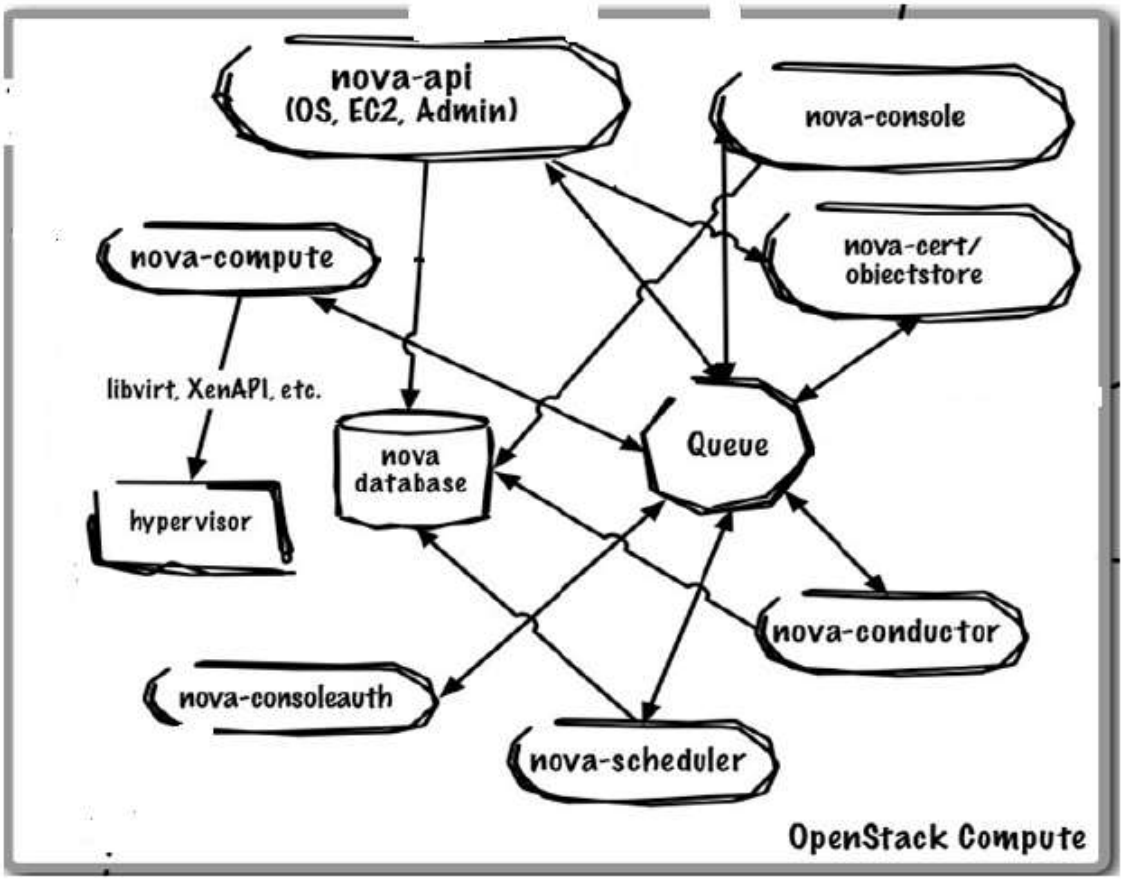


Ilustración 38: Esquema de procesos Nova Openstack

- El cliente `nova-api` acepta y responde a las llamadas del usuario final.
- La virtualización es administrada por `nova-compute`. Crea/finaliza las instancias a través de la API del hipervisor utilizado.

- El almacenamiento es controlado por `nova-volume` (ahora reemplazado por Cinder). Administra la creación, vinculación y desvinculación de volúmenes de almacenamiento persistente a instancias.
- Las redes son gestionadas por `nova-network` (ahora reemplazado por Neutron). Acepta tareas relacionadas a redes y las ejecuta.
- La planificación es realizada por `nova-schedule`. Toma los pedidos de VMs de la cola y determina dónde debería ejecutarse.
- La cola, por defecto RabbitMQ, es el nodo central para el pasaje de mensajes entre daemons.
- También dispone de una base de datos que almacena la mayoría de los datos de estado de compilación y ejecución.
- `nova-consoleauth, nova-novncproxy, nova-console` permiten a los usuarios acceder a las instancias virtuales a través de un proxy.
- Al crear una instancia deberán seleccionar entre las opciones de configuraciones de recursos virtuales disponibles, llamadas flavors. Luego, pueden agregarse recursos como volúmenes de almacenamiento persistente y direcciones IP públicas. [14]

API

`Nova-api` es la encargada de aceptar y responder las llamadas del usuario final a la API de `nova-compute`. El demonio `nova-api` soporta la API de Openstack, la API EC2 de Amazon y la API especial de administración (para usuarios con privilegios que realicen tareas administrativas) [15]

`Nova-api-metadata` acepta solicitudes de metadatos de instancias. Este servicio generalmente solo se utiliza cuando se ejecuta en modo multi-host con instalaciones nova-network.

Nova nos permite acceder y administrar el cloud de máquinas virtuales mediante la consola de texto como lo haríamos en GNU/Linux o mediante la consola gráfica. Lo más aconsejable para la administración siempre es la consola de texto, ya que consume mucho menos recursos que Horizon, aunque tiene el inconveniente de que es más difícil de gestionar y se requieren mayores conocimientos. r

Comandos

Lista de instancias, imagen, sabores.	**$ nova list** **$ nova image-list** **$ nova flavor-list**
Lanzar una instancia usando el nombre de una imagen y de un sabor.	$ nova boot --image IMAGE --flavor FLAVOR nombre_instancia
Ver detalles de instancias	$ nova show nombre_instancia
Ver el log de una instancia	$ nova console-log nombre_instancia
Asignar metadatos a una instancia	$ nova meta volume_instancia set newmeta='my meta data'
Crear un snapshot	$ nova image-create volume_instancia snapshot_instancia $ nova image-show snapshot_instancia
Pause	$ nova pause nombre_instancia $ nova pause snapshot_instancia
Quitar Pausa	$ nova unpause nombre_instancia
Suspender	$ nova nombre_instancia suspend
Unsuspend	$ nova resume nombre_instancia
Parar	$ nova stop nombre_instancia
Empezar	$ nova start nombre_instancia
Reanudar	$ nova rescue nombre_instancia
Redimensionar	$ nova resize nombre_instancia FLAVOR
Reconstruir	$ nova rebuild nombre_instancia IMAGE
Reiniciar	$ nova reboot nombre_instancia

Introducir datos de usuario y archivos en una instancia
O por querer lanzar un script post-install
	$ nova boot --user-data FILE nombre_instancia
	$ nova boot --user-data userdata.txt --image IMAGEN \ --flavor SABOR
	Para validar que el archivo está allí, ssh en la instancia y mirar el fichero /var/lib/cloud.

Crear par de claves	$ nova keypair-add clave_publica > clave_publica.pem $ chmod 600 clave_publica.pem
Iniciar con par de claves asociado	$ nova boot --image IMAGEN --flavor SABOR \ --key_name clave_publica.pem nombre_instancia
Usar SSH para conectarse a una instancia.	ssh -i clave_publica.pem root@servidor

Tabla 3: Comandos Nova Openstack

- ***NOTA**: En Anexo.V se describe el proceso de instalación del cliente en nova en uno de los servidores y un ejemplo de lanzar una instancia.*

5. RESULTADOS

El resultado de todo el proceso ha sido la construcción de dicho sistema nombrado, pero en el camino se han ido solventando problemas con sub-resultados satisfactorios y decisivos, con los cuales se ha ido reconduciendo el proyecto buscando la mejor manera de desarrollar cada uno de los escenario sin dejar atrás en ningún momento el objetivo: creación de un sistema de autoescalado de MongoDB corriendo sobre Openstack.

Se comienza con un planteamiento de campos diferenciados en los que se va a trabajar. Estos son:

- Openstack
- MongoDB
- Monitoreo / Alarma

Con estos tres campos de trabajo se elabora un cuestionario inicial de cada una de las partes, exponiendo qué carencia de conocimientos tenemos y qué conocimientos debemos adquirir.
Se parte de un desconocimiento de cada una de las tecnologías, así como de la metodología que implica el proceso de instalación, puesta en marcha, administración y una carencia de conocimientos base de sistema UNIX, S.O., y comandos de consola.

Este primer planteamiento nos dará idea de dónde debemos profundizar y qué debemos conocer y manejar:

1. Peticiones Openstack: ¿Cuál es el funcionamiento de Openstack en cuando a peticiones que se realizan entre proyectos para un funcionamiento de tareas cotidianas? Resultado: Saber el funcionamiento de API y tokens.
2. Crecimiento MongoDB: ¿Cuál es la mejor manera de ampliar la base de datos siguiendo la bondad de crecimiento horizontal? Resultado: Método correcto con el cual añadir nodos.
3. Monitoreo y Alarma: ¿En qué característica de la base de datos vamos a poner foco para dicho monitoreo: carga usuarios, espacio de datos, herramientas mongo? ¿Cómo se va a desarrollar este sistema de monitoreo, que lenguaje de programación vamos a utilizar? Resultado: Decisión de qué vamos a monitorizar, con qué herramienta: comandos sistema o herramientas mongo; y

cómo vamos a desarrollar el sistema conjunto de esta monitorización junto con la alarma.

Para tratar cada uno de los puntos, creamos tres líneas paralelas de estudio que irán confluyendo a medida que se adquieran conocimientos. Para desarrollar todo el proceso llevado a cabo en el proyecto y adelantándonos a los sucesos, vamos a comenzar describiendo punto por punto los pasos dados.

Posterior a unos conocimientos básicos y necesarios de cada una de las partes del trabajo (qué es Openstack y un rápido vistazo al Dashboard, qué es MongoDB y los métodos funcionamiento, y el más importante en el punto inicial del proyecto: curso avanzado de Linux), se da comienzo al proyecto en cuanto a la parte práctica se refiere:

- Se proporciona usuario y contraseña de Openstack el cual tiene habilitado un área de trabajo llamado: *TFM_MongoDB* con un pool de recursos asignado.
- Se crea la primera máquina llamada MongoDB y se instala (*Apartado 4.3.1.)* la base de datos.
- Se hacen distintas pruebas conociendo en profundidad cada uno de los directorios y qué contiene cada uno.
- Se dedican unos días a la familiarización de la base de datos: acceder a la Shell mongo, crear distintas bases de datos y dentro de ellas colecciones diferentes, crear algún pequeño script (.js) para la inserción de documentos, utilizar comando de mongo enfocando siempre el conocimiento al paso siguiente, acceder a mongo-shell desde consola y lanzar comandos sencillos, etc.

Tras esta primera familiarización pasamos a decidir qué variable vamos a monitorizar. Como primera toma de contacto, se trabaja conjuntamente con la carga de usuarios, es decir, las conexiones a la base de datos; y también con el storage de la misma. Después de días de trabajo y pruebas de monitorización utilizando distintas herramientas, se decide, dada la gran variedad de herramientas MongoDB enfocadas al storage, escoger para monitorizar el almacenamiento de la base de datos. ***Se propone, como punto de ampliación del proyecto, estudiar la variable de carga de usuarios y crear un sistema de autoescalado que lo contemple también, así como cualquier otra variable que pueda influir en el funcionamiento de la base de datos.***

El estudio del almacenamiento de la base de datos consiste en conocer en profundidad qué tiene en cuenta MongoDB a la hora de contabilizar este storage. Este estudio se refleja en el *Apartado 4.3.3. y 4.3.4.* con un resultado claro: el almacenamiento ha de contemplarse teniendo en cuenta los archivos de datos ubicados en `/dbpath` que contienen documentos, índices y extensiones; reflejado todo en una salida de comando denominada f*ileSize.* Además, en el `/dbpath` se generan otro tipo de archivos, como son namespace y journal, este último si se decide utilizarlo. El namespace no suele ocupar mucho espacio, ya que por nombre son 120byte y el journal se elimina cuando MongoDB aplica todas las operaciones de escritura a los archivos de datos (tenemos en cuenta que éste genera archivos de 1Gb o si establece de 128Mb). Este se tomará de referencia en la imposición del punto crítico de alarma dentro del monitoreo.

Para continuar con MongoDB, una vez claro qué vamos a monitorizar y la variable que debemos tener en cuenta hasta el final del proyecto, pasamos a estudiar la manera de aumentar recursos en la base de datos, una vez que la alarma que vamos a implementar nos avise de que el storage está llegando a su máxima capacidad. Comenzamos pues a profundizar en cada uno de los métodos de MongoDB y sobre todo en esa capacidad de crecimiento horizontal que tanto se alaga. El escalado horizontal hace referencia a Sharding, una distribución uniforme de datos por distintos Shard que pueden ser instancias en standalone o ser conjuntos de Replicas-Sets. Como queremos que nuestra infraestructura se asemeje lo máximo posible a un entorno de producción real, decidimos que los Shards van a configurarse como Replica-Set cambiando un planteamiento simple de añadir nodos, a un planteamiento más complejo que es añadir una infraestructura en alta disponibilidad (podría ser alta disponibilidad geográfica si disponemos de infraestructura cloud distribuida) con una accesibilidad continua de los datos. Ambas configuraciones se pueden ver en *Anexo. III* y *Anexo IV*.

Una vez que tenemos nuestra base de datos funcionando en Cluster - Sharding, tenemos que dar marcha atrás para ver el impacto que ha provocado la decisión de monitorizar los datos, y la manera de implementar la comparación de dicho storage con el que se dispone en la infraestructura de Cluster. Vamos a tener nuestros datos distribuidos por todos los Shards y además la infraestructura va a estar divida en servidores que: almacenan datos, replican datos o únicamente poseen función operacional. La monitorización se consigue a través del servidor que funciona como

router, que ve la infraestructura como un Cluster y a las bases de datos las trata como 'un todo' sin tener en cuenta en qué Shard están almacenadas. Con comandos específicos se puede visualizar la distribución de datos y extraer la ocupación de cada uno de los Shard, comparando así este volumen con el total que tenemos asignado para ese servidor virtual y teniendo siempre en cuenta todo el volumen que ocupa el sistema operativo y directorios varios (que será el valor añadido al valor máximo estimado de llenado del Shard). El valor escogido en este proyecto, toma como referencia el número de Shards y un volumen teórico basado en estudios de las máquinas que lo forman y el sabor de cada una, que se ha determinado idéntico para la ampliación. ***En este punto se propone otra ampliación del proyecto correspondiente a un estudio detallado de la relación proporcional o directamente proporcional de los datos generados por la base de datos y los datos de configuración así como del propio sistema operativo. También puede hacerse una ampliación del sistema de autoescalado en la que pueda optarse por ampliar máquinas con un sabor que opte el usuario o administrativo.***

Pasamos a explicar otra línea de trabajo llevado a cabo paralelamente a las anteriores y que es Openstack a nivel de comandos de consola. Al principio usamos la interfaz gráfica de usuario para la creación de las máquinas de MongoDB. [17]

, pero para la automatización se debe conocer qué proyecto entra en juego en dicha creación, su instalación que se llevará a cabo en una de las máquinas de MongoDB, como todos sus comandos de administración. El proyecto es Nova, y la instalación de la parte cliente y comandos que maneja, pueden verse en *Apartado 4.6* y *Anexo V.*

Tanto la parte de monitorización, como de alarma, creación de nuevas máquinas (en Nova) y de iniciación de las mismas como Réplica-Set y posterior inserción al Cluster (en MongoDB), se desarrollan dentro de un script escrito lenguaje de programación *Bash,* que será ejecutado en el servidor principal de Sharding: `mongos` (router). Para la realización de dicho script, primeramente se elabora un pseudocódigo que nos ayudará a tener una idea de la globalidad del sistema, del cuerpo del script y de cada una de las funciones.

- ***Nota:*** *Se anexa el pseudocódigo en Anexo VII*
- ***Nota:*** *Se anexa el sistema de autoescalado final como autoscaling.sh en Anexo VIII*

El script final irá dotado de documentación apropiada para su fácil comprensión.

CONCLUSIONS

After completing the project and summarize all the steps along the way, we can say that the process in each of its points has been satisfactory. It has been implemented step by step, a complete and innovative system currently making use of innovative and stable technologies nowadays with booming companies and managers decide to start new projects, as in this work, installation and commissioning. Each of the lines of work provides an enriching knowledge in these times and also very useful for future work or collaborations, both in the field of research and in private business. Each of these technologies have a community behind of people providing knowledge and improving them, meaning with this that are technologies which are continuously growing.

Companies are currently evolving to take hybrid clouds, and every day more of them will be raised using cloud computing technologies as Openstack, to take advantage of some features such as:

- Single management interface for public, private or hybrid environments.
- A fast development when competitors cloud computing.
- Flexible and agile platform.
- CPU, memory, disk and network interfaces virtualized with better efficiency as it will depend on demand.
- Management of live virtual machines, created from images that can easily be managed in a repository.
- And all with a clear idea: full scalability.

In addition, the management of information produced by the mass of data generated today, also makes necessary the use of databases that can be accommodate and process.

Increasingly, these data fail to follow a predetermined scheme, given the desire to save all; for this reason it will be increasingly common owning and using semi-structured database data.

MongoDB excels at managing high-volume geospatial data (big data). It is therefore, a solution to a real problem (managing large volumes of data that meets the need of horizontal scaling), because the main problem of the traditional model of SQL databases is that this is costly in terms performance .

The advantages, in order to make decisions besides the ones that have already been discussed, are :

- Save the data, instead of tables and organizational charts, documents, describing practically are.
- The dynamic database: as they change documents can be added "hot" new columns, new data, relationships, etc.
- The shell and manipulation of data is handled by code much like manner be written with JavaScript, being easy to understand and use.
- Excellent performance the Scalability the inexpensively.
- The Simplicity in building relationships and data queries.

The union of these two complementary technologies is present in many scenarios, to be chosen separately in the enterprise sector: here is the decision to undertake this work. The result of it helps to facilitate the daily work of those who decide to manage their resources OpenStack virtual host platforms and its semi-structured databases like MongoDB data.

As a final and personal conclusion, this project has raised me as a real challenge. After studying Telecommunications Engineering, I´ve missed tackle a practical project which aim at creating and commissioning of a system that could currently have any business, both SMEs and large enterprises. I have learned to cope and solve problems learning new technologies, with the consequent understanding of new concepts, and to study thorough based on searches for network data repositories and forums where people from beginners to real system administrators, present their own problems to help understanding yours, which have helped me to solve many obstacles. Proud of this advance, I end this project expecting continuing this started way and to contribute to another project or challenge.

6. CONCLUSIONES

Tras finalizar el proyecto y resumir todos los pasos dados en el camino, podemos decir que todo el proceso, en cada uno de sus puntos, ha sido satisfactorio. Se ha implementado, paso a paso, un sistema completo y novedoso haciendo uso de innovadoras y estables tecnologías actualmente en auge con las que empresas y administradores deciden comenzar proyectos nuevos afrontando, como en este trabajo, su instalación y puesta en servicio. Cada una de las líneas de trabajo aporta un conocimiento muy enriquecedor en estos tiempos y también muy útil para trabajos futuros o colaboraciones, tanto en el sector de la investigación como en el privado empresarial. Cada una de estas tecnologías tienen por detrás una comunidad de gente aportando conocimiento y mejorándolas, queriendo decir con esto que son tecnologías en continuo crecimiento. [18]

Las empresas actualmente están evolucionando para tener Clouds híbridas, y cada día un mayor número de ellas se plantearán usar tecnologías de Cloud Computing como Openstack, para beneficiarse de algunas características como las siguientes:

- Interfaz única de administración para entornos públicos, privados o híbridos.
- Un rápido desarrollo en cuando a los competidores de Cloud Computing.
- Plataforma flexible y ágil.
- CPU, memoria, discos e interfaces de red virtualizados con mejor eficiencia en el uso, ya que van a depender de la demanda.
- Gestión de máquinas virtuales en caliente, creadas a partir de imágenes que fácilmente pueden ser administradas en un repositorio.
- Y todo ello con una clara idea: escalabilidad total.

Además, el manejo de información producido por la masificación de datos que se generan actualmente, hace también necesaria la utilización de bases de datos que los puedan albergar y también procesar. Cada vez más, estos datos dejan de seguir un esquema predeterminado, dado el afán de guardarlo todo; por esta razón se hará cada vez más habitual el poseer y utilizar bases de datos semi-estructuradas.
MongoDB destaca en la gestión de datos geoespaciales de gran volumen (big data). Es, por lo tanto, una solución a un problema real (la gestión de grandes volúmenes de datos que responde a la necesidad de escalado horizontal), porque el principal problema del modelo tradicional de bases de datos SQL es que todo esto es costoso

en términos de rendimiento. Las ventajas, a la hora de la toma de decisión aparte de las comentadas, son: [19]

- Guardar los datos, en lugar de tablas y esquemas organizativos, en documentos, describiéndolos prácticamente como son.
- Base de datos dinámica: conforme cambian los documentos se pueden añadir "en caliente" nuevas columnas, nuevos datos, relaciones, etc..
- El shell y la manipulación de los datos se manejan a través de código de manera muy similar a como se escribiría con javascript, siendo fácil de entender y usar.
- Excelente rendimiento
- Escalabilidad sin excesivo coste.
- Sencillez en creación de relaciones y *queries* de datos.

La unión de estas dos tecnologías complementarias se hace presente en bastantes escenarios, por ser elegidas por separado en el sector empresa: he ahí la decisión de emprender este trabajo. El resultado del mismo contribuye a facilitar la tarea diaria de quienes se deciden a gestionar sus recursos virtuales con plataformas Openstack y albergar sus datos semi-estructurados en bases de datos como MongoDB.

Como una conclusión final y personal, este proyecto se me ha planteado como un verdadero reto. Tras estudiar la Ingeniería Técnica de Telecomunicaciones, echaba en falta afrontar un proyecto práctico donde tener como objetivo la creación y puesta en servicio de un sistema que podría tener actualmente cualquier empresa, tanto pyme como gran empresa. He aprendido a afrontar y solventar problemas aprendiendo tecnologías nuevas, con la consecuente comprensión de nuevos conceptos, así como a realizar un estudio profundo basado en búsquedas por la red en repositorios de información y en foros donde personas, desde principiantes hasta verdaderos administradores de sistemas, exponen sus propios problemas que ayudan a la comprensión del tuyo mismo, y que me han ayudado a resolver bastantes obstáculos. Orgullosa del avance, concluyo dicho proyecto esperando proseguir este camino comenzado y poder contribuir con ello a algún otro proyecto o reto.

7. REFERENCIAS

[1] NIST – National Institute of Standars and Technology. "*Definition Cloud Computing".* Fuente: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[2] Buyya Rajkumar, J. B. "*Cloud Computing - Principles and Paradigms*". Fuente: http://www.cloudscaling.com/resources/ (2011)

[3] Red Hat, "*Qué es Openstack*". Fuente: https://www.redhat.com/es/insights/openstack

[4] Maish Saidel-Keesing, "*Tendencias y predicciones de virtualización de servidores para 2015*". Fuente: http:// searchdatacenter.techtarget.com /

[5] ABM Moniruzzaman, Syed Akhter Hossain. "*NoSQL Database: New Era of atabases for Big data Analytics - Classification, Characteristics and Comparison"*. 30 Jun 2013, Department of Computer Science and Engineering Daffodil International University.

[6] Mongo University, "*Course M102: MongoDB for DBAs*". Fuente: https://university.mongodb.com/

[7] C. Creative, "*Binary JSON*". Fecha: 2012, Fuente: http://bsonspec.org

[8] MongoDB Cloud Manager. Fuente: mms.mongodb.com/user/login

[9] Documentation Replication. Fuente: http://docs.mongodb.org/manual/core/replication/

[10] Documentation Sharding. Fuente: http://docs.mongodb.org/manual/core/sharding/

[11] T. Brock, "*Sharding*" Presentation, 2013. Available:

http://www.10gen.com/presentations/sharding-1

[12] Kristina Chodorow, "*Scaling MongoDB*", Editor: Mike Loukides (February 2011)

[13] Opentack, "*Chapter 5. Compute Node*". Fuente: http://docs.openstack.org/icehouse/training-guides/content/associate-computer-node.html

[14] Openstack, "*Command-line Utilities NOVA*". Fuente: http://docs.openstack.org/developer/nova/man/index.html

[15] Opentack, "*Compute API v2*", http://developer.openstack.org/api-ref-compute-v2.html

[16] "*OpenStack Nova Client*", IES Gonzalo Nazareno Dos Hermanas (Sevilla) IES Los Albares Cieza (Murcia) IES La Campi˜na Arahal (Sevilla) IES Ingeniero de la Cierva Murcia.

[17] S. Ramakrishnan**,** *"An Efficient Failover Enabling Mechanism In Openstack".* Cloud Computing Department of Information Technology SRM University, SRM Nagar, Kattankulathur, Tamil Nadu, India, 2013.

[18] Adriana E. Martín, Susana Chávez, María Murazzo, Nelson Rodríguez, Adriana Valenzuela5, "*MongoDB en ambiente Cloud Híbrido con OpenStack*". Departamento e Instituto de Informática - F.C.E.F. y N. - U.N.S.J

[19] [E. Gunderson, "Two reasons you souldn't use MongoDB," Online, 2010. [Online]. Available: http://ethangunderson.com/blog/two-reasons-to-not-use-mongodb/.]

Resolución problemas teóricos y prácticos MongoDB y Opentack,. Fuente: http://stackoverflow.com/questions

Resolución problemas teóricos y prácticos MongoDB y Opentack. Fuente: https://gist.github.com

8. ANEXOS

Se anexas toda la documentación referente a código usada en el desarrollo del proyecto.

Anexo.I INSERCCIÓN DE DATOS MONGODB

Para la inserción de datos se ha creado una base de datos “maria” donde a través consola, se ejecutará un fichero JavaScript que albergará comandos para que, mediante un bucle, se vaya llenando la DB.

Fichero .js

```
var t=db.insercion;   //inserción será la colección

if( t.count() ) {
print("Ya existe la base de datos. Se procede a borrar la existente.");
t.drop();
    }  //como vamos a hacer varias pruebas, lo conveniente es borrar colecciones

var a = 0;

for( var m = 0; m < 1000000 ; m++ ) {
        var ts = new Date(2015,m%12,m%27+1);
        t.insert({ _id : m, tstamp : ts, active : (a%77==0), x : 99 } );
        a += 3;
}
printjson( db.getLastErrorObj() ); // Es una función para visualizar
op.de escritura
print("Cargando...");
// Esto se añade para futuras pruebas
//t.update({},{$set:{str:"this is a test"}},false,true);
//printjson( db.getLastErrorObj() );
print( "count: " + t.count() );
```

¿Cómo haremos la inserción desde consola?

```
[root@mongodb ~]# mongo maria < insercion datos.js
connecting to: maria
Ya existe la base de datos. Procedemos a borrar la existente;
true
1000000
{
        "connectionId" : 235,
```

```
        "n" : 0,   // nº de documentos actualizados
        "syncMillis" : 0,
        "writtenTo" : null,
        "err" : null,
        "ok" : 1
}
cargando...
count: 1000000
```

Como puede verse, se ha hecho una inserción a través de un .js y desde la consola cargando dicho fichero a la DB, según vimos en el apartado 6.

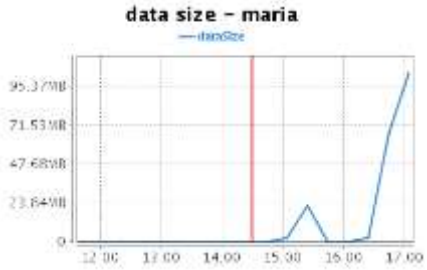
Anexo.II RESULTADOS DE MONITORIZACIÓN DEL ALMACENAMIENTO DE MONGODB

Tras la inserción de la colección, vemos el estado del almacenamiento o *storage* utilizando primeramente el comando mongo `dbstats` y contrastamos dichos resultados con los datos gráficos que podemos obtener del servicio de monitorización de Mongo MMS:

```
        "db" : "maria",
        "collections" : 4,
        "objects" : 968242,
        "avgObjSize" :
111.9996034049339,
        "dataSize" : 108442720,
        "storageSize" : 123961344,
        "numExtents" : 14,
        "indexes" : 2,
        "indexSize" : 27046208,
        "fileSize" : 201326592,
        "nsSizeMB" : 16,
        "dataFileVersion" : {
          "major" : 4, "minor" : 5},
        "extentFreeList" : {
           "num" : 0, "totalSize" :
0},
        "ok" : 1
```

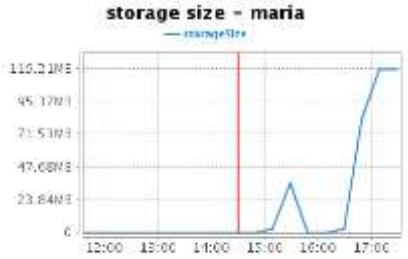
```
      "db" : "maria",
        "collections" : 3,
        "objects" : 5,
        "avgObjSize" : 60.8,
        "dataSize" : 304,
        "storageSize" : 24576,
        "numExtents" : 3,
        "indexes" : 1,
        "indexSize" : 8176,
        "fileSize" : 134217728,
        "nsSizeMB" : 16,
        "dataFileVersion" : {
           "major" : 4, "minor" : 5},
        "extentFreeList" : {
           "num" : 14, "totalSize" :
8090828 },
        "ok" : 1
```


TFM_Mongo
mongodb:27017
May 13, 2015 5:35PM CEST
avg / sec
MMS by mongoDB
avg obj size - maria
collections - maria
data size - maria
file size - maria
index size - maria
indexes - maria
lock % - maria
num extents - maria
objects - maria
storage size - maria

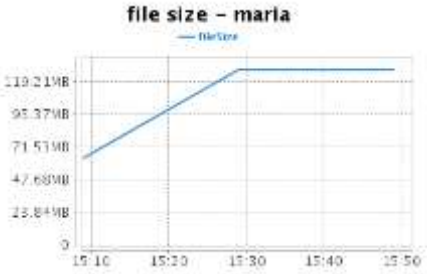


Como podemos ver, tanto los datos de `dbstats` como los MMS son similares.

"dataSize":108442720Bytes. Se puede ver que el tamaño asignado a la DB era casi nulo, aumentando el total del tamaño de la DB.



"storageSize":123961344Bytes dataSize+ extension. Hay 14,8MB de diferencia que se corresponden al espacio aún no utilizado por las extensiones (que como podemos ver se han creado 14).



"fileSize" : 134217728 Byte. 134Mb que corresponde al storageSize más el espacio de almacenamiento en el disco. Es mayor que storageSize porque incluye extensiones de índice y espacio todavía-no utilizado en los archivos de datos.

Vamos a ver qué relación tiene la señalada DB a la que estamos haciendo referencia con el storage total asignado como MV.

- Buscamos el directorio donde están almacenados los archivos de datos de la DB creadas.

```
/var/lib/mongo/
[root@mongodb mongo]# ls -la
total 409648
drwxr-xr-x.  3 mongod mongod     4096 May 14 08:30 .
drwxr-xr-x. 21 root   root       4096 Mar  1 19:19 ..
-rw-------   1 mongod mongod 16777216 May 11 20:09 admin.0
-rw-------   1 mongod mongod 16777216 May  6 19:15 admin.ns
drwxr-xr-x.  2 mongod mongod     4096 May 14 08:30 journal
-rw-------.  1 mongod mongod 67108864 May 14 08:30 local.0
-rw-------.  1 mongod mongod 16777216 May 14 08:30 local.ns
-rw-------   1 mongod mongod 67108864 May 13 16:52 maria.0
-rw-------   1 mongod mongod 67108864 May 13 16:51 maria.1
-rw-------   1 mongod mongod 67108864 May 13 16:52 maria.2
-rw-------   1 mongod mongod 16777216 May 13 16:52 maria.ns
-rwxr-xr-x.  1 mongod mongod        5 May 14 08:30 mongod.lock
-rw-------   1 mongod mongod 67108864 May 12 19:38 test.0
-rw-------   1 mongod mongod 16777216 May 12 19:38 test.ns
```

- Cogemos los 3 archivos que se han creado para la DB maria y calculamos el total de Mb utilizados:

```
ls -la | grep maria | awk '{print $9"\t"$5/1048576"Mb"}' |awk
'{suma+=$2}END{ print suma }'
[root@mongodb mongo]# ls -la | grep maria | awk '{print
$9"\t"$5/1048576"Mb"}'
maria.0   64Mb
maria.1   64Mb
maria.2   64Mb
maria.ns  16Mb

[root@mongodb mongo]# ls -la | grep maria | awk '{print
$9"\t"$5/1048576"Mb"}' |awk '{suma+=$2}END{ print suma" Mb totales"}'
208 Mb totales
```

- Contractamos con la info de la shell de mongo:

```
[root@mongodb mongo]# mongo
MongoDB shell version: 2.6.4
connecting to: test
> show dbs
admin   0.031GB
config  (empty)
local   0.078GB
maria   0.203GB
test    0.078GB
```

- Ahora vamos a coger la info que nos dan los comandos del sistema.
  Primero vamos a utilizar el comando `df` para ver el espacio total, usado y libre de la partición asignada.

```
[root@mongodb /]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1       9.9G  5.2G  4.2G  56% /
tmpfs           939M     0  939M   0% /dev/shm
```

- Ahora, una vez que tenemos una visión del espacio utilizado, vamos a desgranar de qué directorios proviene.

```
[root@mongodb /]# du -ch --max-depth=1
3.5G    ./var
4.0K    ./cgroup 5.0G    total
```

Si hacemos un filtrado para mostrar los directorios relacionados con mongo y su volumen:

```
[root@mongodb /]# du –max-depth=3| sort -nr -k1 | grep mongo
1720376 ./var/lib/mongo
1493952 ./var/log/mongodb
186464  ./var/log/mongodb-mms-automation
51620   ./var/lib/mongodb-mms-automation
46500   ./opt/mongodb-mms-automation
46488   ./opt/mongodb-mms-automation/versions
8       ./var/run/mongodb-mms-automation
8       ./var/run/mongodb
8       ./opt/mongodb-mms-automation/bin
8       ./etc/mongodb-mms
```

Comprobamos que el directorio con más storage utilizado es `/var`, al igual que la información general obtenida. Esto nos da una idea de que un servidor con un S.O. y sobre él una DB no solo utiliza espacio de la propia DB, sino también espacio de sistema, y es aquí donde cada usuario debe determinar qué porcentaje utilizará para hacer saltar la alarma por estar al borde de la saturación de espacio.

Anexo.III REPLICA SET

Primero convertimos un nodo standalone en miembro del RS. Para ello, comenzamos con una parada del servicio:

```
[root@mongodb log]# /etc/init.d/mongod stop
Stopping mongod:                                           [  OK  ]
[root@mongodb log]# ps aux | grep mongo
root     17984  0.0  0.0 103244   848 pts/0    S+   11:20   0:00 grep
mongo
```

A continuación modificamos las carpetas para `datadb` y `log` en los directorios donde en la instalación veíamos que MongoBD almacenaba ambos datos. También cambiamos el `hostname` (*Importante*! Cambiarlo en el directorio /etc/hosts), que nos facilitará la nomenclatura a la hora de crear más RS y también a la hora de manejar dicho RS añadiendo más nodos.

```
[root@mongodb log]# mv /var/lib/mongo /var/lib/mongo0
[root@mongodb log]# vi /var/log/mongodb/log0.log     #El log para el
standalone era mongod.log, creamos otro archivo mejor.
[root@mongodb log]# hostname mongodb-0     #vi /etc/hosts
[root@mongodb mongodb]# mongod --port 27017 --dbpath /var/lib/mongo0 --
logpath /var/log/mongodb/log0.log --smallfiles --logappend --oplogSize
50 --replSet rs0
#podemos ejecutar el comando con toda la configuración o llamar al archivo /etc/mongd.conf
y llamarlo tras mongod con un -f

[root@mongodb mongodb]# mongo
MongoDB shell version: 2.6.4
connecting to: test
> rs.initiate()
{
        "info2" : "no configuration explicitly specified -- making one",
        "me" : "mongodb0:27017",
        "ok" : 0,
        "errmsg" : "couldn't initiate : can't find self in the replset
config"
}
> rs.conf()
```

```
null
```

Salimos de la Shell a la consola y vemos qué ha ocurrido en el log.

```
2015-05-31T12:37:04.423+0200 [rsStart] replSet can't get
local.system.replset config from self or any seed (EMPTYCONFIG)
2015-05-31T12:37:04.424+0200 [rsStart] replSet info you may need to run
replSetInitiate -- rs.initiate() in the shell -- if that is not already
done
```

Volvemos a la Shell y configuramos el Replica Set.

```
> var rsconfig = {
"_id":"rs0",
members:[ {"_id":0, host:"mongodb0:27017"} ]
}
> print(JSON.stringify(rsconfig))
{"_id":"rs0","members":[{"_id":0,"host":"mongodb0:27017"}]}
>
> rs.initiate(rsconfig)
{
        "info":”Config now saved locally. Should come online in about a
minute.",
        "ok" : 1
}
rs0:PRIMARY> rs.status()
{
        "set" : "rs0",
        "date" : ISODate("2015-05-31T11:42:24Z"),
        "myState" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "name" : "mongodb0:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 691,
                        "optime" : Timestamp(1433072501, 1),
                        "optimeDate" : ISODate("2015-05-31T11:41:41Z"),
                        "electionTime" : Timestamp(1433072502, 1),
                        "electionDate": ISODate("2015-05-31T11:41:42Z"),
                        "self" : true
```

```
                }
        ],
        "ok" : 1
}
```

Una vez que todo está OK para el Primario, añadimos dos miembros, que es el número mínimo al RS. Vamos a hacerlo con máquinas virtuales diferentes. Para ello, instalamos en cada una de ellas MongoDB y configuramos el fichero `/etc/mongo.conf` como explicamos en paso anteriores:

```
Hostname mongodb1
/var/lib/mongo1
/var/log/mongodb/log1.log
```

En cada una de las máquinas ejecutamos `mongod` con la configuración de Replica Set, una vez que hemos parado el proceso que debe correr correspondiente al nodo en modo *standalone*.

```
[root@mongodb2 /]# mongod --dbpath /var/lib/mongo1 –-logpath
/var/log/mongodb/log1.log --logappend --port 27018 --oplogSize 50 --
smallfiles --fork --replSet rs0

[root@mongodb2 /]# mongod --dbpath /var/lib/mongo2 --logpath
/var/log/mongodb/log2.log --logappend --port 27019 --oplogSize 50 --
smallfiles --fork --replSet rs0
```

Nos conectamos de nuevo al primario para cargar la configuración de los 2 secundarios que vamos a añadir al RS.

```
rs0:PRIMARY> var cfg = rs.conf()
rs0:PRIMARY> cfg
{
        "_id" : "rs0",
        "version" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "mongodb0:27017"
                }
        ]
}
rs0:PRIMARY> cfg.members[1] = {"_id":1,"host":"10.10.0.19:27018"}
```

```
{ "_id" : 1, "host" : "10.10.0.19:27018" }
rs0:PRIMARY> cfg.members[2] = {"_id":2,"host":"10.10.0.20:27019"}
{ "_id" : 2, "host" : "10.10.0.20:27019" }
rs0:PRIMARY> rs.reconfig(cfg)
{ "ok" : 1 }
rs0:PRIMARY> rs.status()
{
        "set" : "rs0",
        "date" : ISODate("2015-06-04T10:55:25Z"),
        "myState" : 1,
        "members" : [
           {
                   "_id" : 0,
                   "name" : "mongodb-0:27017",
                   "health" : 1,
                   "state" : 1,
                   "stateStr" : "PRIMARY",
                   "uptime" : 7420,
                   "optime" : Timestamp(1433415310, 1),
                   "optimeDate" : ISODate("2015-06-04T10:55:10Z"),
                   "electionTime" : Timestamp(1433412721, 2),
                   "electionDate": ISODate("2015-06-04T10:12:01Z"),
                   "self" : true
           },
           {
                   "_id" : 1,
                   "name" : "10.10.0.19:27018",
                   "health" : 1,
                   "state" : 5,
                   "stateStr" : "STARTUP2",
                   "uptime" : 15,
                   "optime" : Timestamp(0, 0),
                   "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                   "lastHeartbeat":ISODate("2015-06-04T10:55:24Z"),
                   "lastHeartbeatRecv":ISODate("2015-06-04T10:55:23Z"),
                   "pingMs" : 0,
                   "lastHeartbeatMessage" : "initial sync need a member
to   be primary or secondary to do our initial sync"
           },
           {
                   "_id" : 2,
                   "name" : "10.10.0.20:27019",
                   "health" : 1,
                   "state" : 5,
                   "stateStr" : "STARTUP2",
                   "uptime" : 15,
```

```
                        "optime" : Timestamp(0, 0),
                        "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                        "lastHeartbeat:ISODate("2015-06-04T10:55:24Z"),
                        "lastHeartbeatRecv":ISODate("2015-06-04T10:55:23Z"),
                        "pingMs" : 17,
                        "lastHeartbeatMessage" : "initial sync need a member
to   be primary or secondary to do our initial sync"
                    }
                ],
                "ok": 1
```

Una vez que tenemos la configuración, echamos un vistazo al `oplog` en cada uno de los nodos secundarios.

```
rs0:SECONDARY> use local
switched to db local
rs0:SECONDARY> db.oplog.rs.find()
{ "ts" : Timestamp(1433415310, 1), "h" : NumberLong("-
6020442515650379759"), "v" : 2, "op" : "n", "ns" : "", "o" : { "msg" :
"Reconfig set", "version" : 2 } }

rs0:SECONDARY> db.oplog.rs.stats()
{
        "ns" : "local.oplog.rs",
        "count" : 1,
        "size" : 100,
        "avgObjSize" : 100,
        "storageSize" : 52428800,
        "numExtents" : 1,
        "nindexes" : 0,
        "lastExtentSize" : 52428800,
        "paddingFactor" : 1, # porque es una capped collections
        "systemFlags" : 0,
        "userFlags" : 0,
        "totalIndexSize" : 0,
        "indexSizes" : {

        },
        "capped" : true, # si, a 50Gb
        "max" : NumberLong("9223372036854775807"),
        "ok" : 1
}
```

Anexo.IV SHARDING

Para llegar a completar el Cluster Sharding, tenemos que adecuar la infraestructura ya montada del punto anterior, y seguir unos cuanto pasos.

Preparar Replica-Set para ser Replica-Set del Cluster Sharding

Convertimos nuestro Replica-Set en un Replica-Set del Cluster de Shard. Debemos cambiar la sintaxis utilizada en el montaje de dicho Replica Set para no confundirnos una vez tengamos toda la arquitectura de desarrollo. Son buenas prácticas para después no cometer fallos.

Comenzamos pues, un proceso de cambio de la configuración del Replica Set. Los pasos son:

1. Parar el servicio en todos los miembros.
2. Iniciar cada uno de ellos en un puerto diferente como standalone. Antes, debemos haber hecho los cambios (por ejemplo como es este caso) en los nombres de carpetas, para que el `--dbpath` sea el correcto ya en la nueva configuración.
3. Ya con el `mongod` inicializado, nos metemos en la Shell y editamos desde aquí el Replica Set, usando el comando `db.system.replset.findOne` para encontrar la configuración por el identificativo de RS.
4. Paramos los procesos e inicializamos de nuevo el `mongod` en el puerto nuevo y con la opción de `--replSet`.
5. En cada instancia nos conectamos a la Shell y confirmamos la nueva configuración llamando a `rs.conf()`

```
[root@mongodb-0 ~]# /etc/init.d/mongod stop
Stopping mongod:                                           [  OK  ]

[root@mongodb-0 ~]# mongod --port 27055 --dbpath /var/lib/mongor00

[root@mongodb-0 ~]# mongo --port 27055
MongoDB shell version: 2.6.4
connecting to: 127.0.0.1:27055/test
> use local
switched to db local
> cfg = db.system.replset.findOne({_id:"rs0"})
{
        "_id" : "rs0",
        "version" : 2,
```

```
        "members" : [
                {
                        "_id" : 0,
                        "host" : "mongodb-0:27017"
                },
                {
                        "_id" : 1,
                        "host" : "10.10.0.19:27018"
                },
                {
                        "_id" : 2,
                        "host" : "10.10.0.20:27019"
                }
        ]
}
> cfg.members[0].host = "10.10.0.18:27000"
10.10.0.18:27000
> cfg.members[1].host = "10.10.0.19:27001"
10.10.0.19:27001
> cfg.members[2].host = "10.10.0.20:27002"
10.10.0.20:27002
> cfg
{
        "_id" : "rs0",
        "version" : 2,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "10.10.0.18:27000"
                },
                {
                        "_id" : 1,
                        "host" : "10.10.0.19:27001"
                },
                {
                        "_id" : 2,
                        "host" : "10.10.0.20:27002"
                }
        ]
}
> db.system.replset.update({_id:"rs0"}, cfg)
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
[root@mongodb-0 ~]# mongod -f /etc/mongod.conf   # en dicho archivo
tenemos cargado la configuración con: --dbpath --logpath --fork --
smallfiles --replSet r00 --oplogSize 50 --port 27000

[root@mongodb-0 ~]# mongo --port 27000
MongoDB shell version: 2.6.4
connecting to: 127.0.0.1:27000/test
rs0:PRIMARY>
rs0:PRIMARY> rs.conf()
{
        "_id" : "rs0",
        "version" : 2,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "10.10.0.18:27000"
                },
                {
                        "_id" : 1,
                        "host" : "10.10.0.19:27001"
                },
                {
                        "_id" : 2,
                        "host" : "10.10.0.20:27002"
                }
        ]
}
```

*Este último paso se repite para los 2 miembros secundarios.

Creación del Cluster de Sharding

El paso siguiente es convertir nuestro RS en Cluster de Sharding, ya que partimos de este escenario, y añadir los demás miembros.

El escenario de Sharding que vamos a poner en marcha hará un total de 10 máquinas virtuales:

- ✓ Tres servidores de configuración, cada uno en una máquina diferente.
- ✓ Un servidor para la instancia mongos.
- ✓ 2 Replica-Set de 3 miembros cada uno.

El procedimiento es el siguiente:

1. Iniciar cada miembro del RS (inicializados del paso anterior).
   Los 3 host son:

10.10.0.18 27000 → PRIMARIO

10.10.0.19 27001 → SECUNDARIO

10.10.2.20 27002 → SECUNDARIO

2. Crear 3 máquinas para los 3 servidores de configuración. Dichas máquinas serán:
   10.10.0.21 27100

   10.10.0.22 27101

   10.10.2.23 27102

   La configuración para estas instancias es la siguiente:

```
configsvr=true
port=27100 / 27101 / 27102
dbpath=/var/lib/mongocfg0/1/2
logpath=/var/log/mongodb/cfg0/1/2.log
logappend=true
fork=true
pidfilepath=/var/run/mongodb/mongod.pid
bind_ip=0.0.0.0
```

   Y así inicializamos la instancia `mongo` desde el archivo de configuración con -f más cómodamente (hay que borrar mongod.lock de /var/lib/mongocfg porque da error al ejecutar la instancia).

```
[root@s-cfg0 ~]# mongod -f /etc/mongod.conf
```

   Importante!!! Actualizar el fichero /etc/hosts con todos los nombres de las máquinas que vayamos creando, en las de RS, en las de servidores de configuración y en el router mongos.

3. Instancia `mongos`
   Creamos otra máquina y la configuramos para inicializar nuestra instancia `mongos`.

```
[root@router-mongos ~]# vi /etc/mongod.conf
port=27017
logpath=/var/log/mongodb/mongos.log
logappend=true
fork=true
```

```
pidfilepath=/var/run/mongodb/mongod.pid
bind_ip=0.0.0.0
(chunk por defecto 64Mb)
[root@router-mongos ~]# mongos --configdb
10.10.0.21:27100,10.10.0.22:27101,10.10.0.23:27102 --logpath
/var/log/mongodb/mongo
```

4. Añadimos el Replica-Set al Shard. Para añadir el RS nos conectamos a la Shell del `mongos` y lo añadimos con el comando `sh.addShard()`

```
[root@router-mongos ~]# mongo 10.10.0.24:27017/admin
mongos>
sh.addShard("rs0/10.10.0.18:27000,10.10.0.19:27001,10.10.0.20:27002")
{ "shardAdded" : "rs0", "ok" : 1 }
```

5. Por último, y para finalizar nuestra arquitectura, vamos a crear otro Replica-Set ("rs1") desde 0 y lo añadimos al Cluster de Shards como hemos hecho en pasos anteriores para el "rs0".

10.10.0.25 27003 → PRIMARIO
10.10.0.26 27004 → SECUNDARIO
10.10.0.27 27005 → SECUNDARIO

```
# Configuración del Replica Set
> var rsconfig = { "_id":"rs1",
members:[{"_id":3,"host":"10.10.0.25:27003"}] }
> rs.initiate(rsconfig)
{
        "info" : "Config now saved locally.  Should come online in about
a minute.            ",
        "ok" : 1
rs1:PRIMARY> rs.add("10.10.0.26:27004")  # Otra manera de añadir
miembros sin utilizar variables
{ "ok" : 1 }
rs1:PRIMARY> rs.add("10.10.0.27:27005")
{ "ok" : 1 }

# Configuración del Resplica-Set dentro del Cluster conectándonos a mongos
```

```
mongos>
sh.addShard("rs1/10.10.0.25:27003,10.10.0.26:27004,10.10.0.27:27005")
{ "shardAdded" : "rs1", "ok" : 1 }
```

Configuración del Cluster de Sharding con chunk

Una vez que ya tenemos la arquitectura, vamos a configurar dicho cluster. Comenzamos conectándonos al Shell de `mongos` para ver el estado de nuestro Cluster de Shards:

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("55716f35d2e1b0bea7f0776f")
}
  shards:
        {  "_id" : "rs0",  "host" :
"rs0/10.10.0.18:27000,10.10.0.19:27001,10.10.0.20:27002" }
        {  "_id" : "rs1",  "host" :
"rs1/10.10.0.25:27003,10.10.0.26:27004,10.10.0.27:27005" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" :
"config" }
        {  "_id" : "maria",  "partitioned" : false,  "primary" : "rs0" }
        {  "_id" : "test",  "partitioned" : false,  "primary" : "rs0" }

mongos> use config
switched to db config

mongos> show dbs
admin   (empty)
config  0.016GB
maria   0.203GB
test    0.078GB
```

```
mongos> show collections
changelog
chunks
databases
lockpings
locks
mongos
settings
shards
system.indexes
versión
#Estas con las colecciones que crea Sharding y que no debemos tocar nunca. Utilizaremos chunk para ver que particiones ha hecho.
```

Vamos a usar la base de datos “maria” para particionar los datos que estaban ya insertados. Para ello usamos el comando `sh.enableSharding()`

```
mongos> sh.enableSharding("maria")
{ "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("55716f35d2e1b0bea7f0776f")
}
  shards:
        {  "_id" : "rs0",  "host" :
"rs0/10.10.0.18:27000,10.10.0.19:27001,10.10.0.20:27002" }
        {  "_id" : "rs1",  "host" :
"rs1/10.10.0.25:27003,10.10.0.26:27004,10.10.0.27:27005" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" :
"config" }
        {  "_id" : "maria",  "partitioned" : true,  "primary" : "rs0" }
        {  "_id" : "test",  "partitioned" : false,  "primary" : "rs0" }
```

Particionamos la colección de la base de datos “maria”: sensor-reading. Lo haremos estableciendo como clave de shard el campo “ id” de dicha colección usando el comando `sh.shardCollection()` con los siguientes parámetros:

`sh.shardCollection(namespace, key, unique)`

```
mongos> sh.shardCollection("maria.sensor_readings", {_id:1}, true)
{ "collectionsharded" : "maria.sensor_readings", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("55716f35d2e1b0bea7f0776f")
}
  shards:

{"_id":"rs0","host":"rs0/10.10.0.18:27000,10.10.0.19:27001,10.10.0.20:27
002" }

{"_id":"rs1","host”:"rs1/10.10.0.25:27003,10.10.0.26:27004,10.10.0.27:27
005" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" :
"config" }
        {  "_id" : "maria",  "partitioned" : true,  "primary" : "rs0" }
                maria.sensor_readings
                        shard key: { "_id" : 1 }
                        chunks:
                                rs1     1
                                rs0     5
                    { "_id" : { "$minKey" : 1 } } -->> { "_id" : 0 } on : rs1
Timestamp(2, 0)
                    { "_id" : 0 } -->> { "_id" : 149796 } on : rs0 Timestamp(2, 2)
                    { "_id" : 149796 } -->> { "_id" : 299593 } on : rs0 Timestamp(2,
3)
                    { "_id" : 299593 } -->> { "_id" : 599187 } on : rs0 Timestamp(1,
1)
                    { "_id" : 599187 } -->> { "_id" : 898781 } on : rs0 Timestamp(1,
2)
```

```
                    { "_id" : 898781 } -->> { "_id" : { "$maxKey" : 1 } } on : rs0
Timestamp(1, 3)
        {  "_id" : "test",  "partitioned" : false,  "primary" : "rs0" }
```

Si analizamos la información, vemos que se han creado 6 chunks (cada uno del máximo establecido por defecto en la configuración: 64Mb) que han correspondido 5 al "rs0" y 1 al "rs1". El rango que ha escogido chunk se puede ver a continuación, que va en función del _id usando la `db config y` la colección chunk como hemos comentado antes:

```
mongos> use config
mongos> db.chunks.find()
{ "_id" : "maria.sensor_readings-_id_MinKey", "lastmod" : Timestamp(2,
0), "lastmodEpoch" : ObjectId("55748f64d2e1b0bea7f0f5a3"), "ns" :
"maria.sensor_readings", "min" : { "_id" : { "$minKey" : 1 } }, "max" :
{ "_id" : 0 }, "shard" : "rs1" }
{ "_id" : "maria.sensor_readings-_id_299593.0", "lastmod" : Timestamp(4,
1), "lastmodEpoch" : ObjectId("55748f64d2e1b0bea7f0f5a3"), "ns" :
"maria.sensor_readings", "min" : { "_id" : 299593 }, "max" : { "_id" :
599187 }, "shard" : "rs0" }
{ "_id" : "maria.sensor_readings-_id_599187.0", "lastmod" : Timestamp(1,
2), "lastmodEpoch" : ObjectId("55748f64d2e1b0bea7f0f5a3"), "ns" :
"maria.sensor_readings", "min" : { "_id" : 599187 }, "max" : { "_id" :
898781 }, "shard" : "rs0" }
{ "_id" : "maria.sensor_readings-_id_898781.0", "lastmod" : Timestamp(1,
3), "lastmodEpoch" : ObjectId("55748f64d2e1b0bea7f0f5a3"), "ns" :
"maria.sensor_readings", "min" : { "_id" : 898781 }, "max" : { "_id" : {
"$maxKey" : 1 } }, "shard" : "rs0" }
{ "_id" : "maria.sensor_readings-_id_0.0", "lastmod" : Timestamp(3, 0),
"lastmodEpoch" : ObjectId("55748f64d2e1b0bea7f0f5a3"), "ns" :
"maria.sensor_readings", "min" : { "_id" : 0 }, "max" : { "_id" : 149796
}, "shard" : "rs1" }
{ "_id" : "maria.sensor_readings-_id_149796.0", "lastmod" : Timestamp(4,
0), "lastmodEpoch" : ObjectId("55748f64d2e1b0bea7f0f5a3"), "ns" :
"maria.sensor_readings", "min" : { "_id" : 149796 }, "max" : { "_id" :
299593 }, "shard" : "rs1" }
```

Para comprobar el correcto funcionamiento de Sharding, vamos a añadir más documentos y ver cómo los va redirigiendo al shard correspondiente.

```
mongos> db.sensor_readings.count()
968234
# Añadimos 10.000 documentos más

mongos> for( var m = 968234; m < 978234; m++ ) {   var a=0; var ts = new
Date(2012,m%12,m%27+1); db.sensor_readings.insert({ _id : m, tstamp :
ts, active : (a%77==0), x : 99 } ); a += 3; }

mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "version" : 4,
        "minCompatibleVersion" : 4,
        "currentVersion" : 5,
        "clusterId" : ObjectId("55716f35d2e1b0bea7f0776f")
}
  shards:

{"_id”:"rs0","host":"rs0/10.10.0.18:27000,10.10.0.19:27001,10.10.0.20:27
002" }
        {
"_id":"rs1","host":"rs1/10.10.0.25:27003,10.10.0.26:27004,10.10.0.27:270
05" }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" :
"config" }
        {  "_id" : "maria",  "partitioned" : true,  "primary" : "rs0" }
                maria.sensor_readings
                        shard key: { "_id" : 1 }
                        chunks:
                                rs1     3
                                rs0     3
                { "_id" : { "$minKey" : 1 } } -->> { "_id" : 0 } on : rs1
Timestamp(2, 0)
                { "_id" : 0 } -->> { "_id" : 149796 } on : rs1 Timestamp(3, 0)
```

```
                        { "_id" : 149796 } -->> { "_id" : 299593 } on : rs1 Timestamp(4,
0)
                        { "_id" : 299593 } -->> { "_id" : 599187 } on : rs0 Timestamp(4,
1)
                        { "_id" : 599187 } -->> { "_id" : 898781 } on : rs0 Timestamp(1,
2)
                        { "_id" : 898781 } -->> { "_id" : { "$maxKey" : 1 } } on : rs0
Timestamp(1, 3)
        {  "_id" : "test",  "partitioned" : false,  "primary" : "rs0" }

mongos>
```

Como puede comprobarse, los datos han sido redirigidos al “rs1” aumentando el número de chunks.

Anexo.V INSTALACIÓN DEL CLIENTE NOVA

Acercándonos al objetivo del proyecto, vamos a instalar un cliente `nova` en el servidor que hemos configurado como router Sharding - `mongos`, ya que lo vamos a utilizar como servidor que va a monitorizar, y desde el cual se van a lanzar las máquinas cuando sea necesario.

Para la instalación de nova, utilizamos los siguientes paquetes con el comando `yum` como hicimos para instalar el servicio MongoDB.

```
yum install http://repos.fedorapeople.org/repos/openstack/openstack-
icehouse/rdo-release-icehouse-3.noarch.rpm
yum install http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-
release-7-5.noarch.rpm
yum install python-novaclient
```

A continuación creamos el directorio `.creds/openrc-mbermejo`, para facilitar el acceso. Mostramos el contenido del fichero, teniendo que rellenar los datos con los que tengamos en la plataforma Openstack (en el apartado 4.2.1. se ve dónde encontrar el contenido de las APIs de los proyectos con los que ha de conectar `nova`).

```
[root@router-mongos ~]# vi .creds/openrc-mbermejo
export OS_USERNAME=mbermejo
export OS_TENANT_NAME=TFM_MongoDB
export OS_PASSWORD= ******
export OS_SERVICE_ENDPOINT=https://controller.ceta-ciemat.es:5000/v2.0
export OS_AUTH_URL=https://controller.ceta-ciemat.es:5000/v2.0/
```

Tras la creación de dicho directorio modificamos `/etc/resolv.conf` para evitar problemas con el DNS:

```
[root@router-mongos ~]# vi /etc/resolv.conf
; generated by /sbin/dhclient-script
search ceta-ciemat.es
nameserver 8.8.8.8
```

Pasamos a instalar el certificado de la root - CA - que firmó el certificado del controller.ceta-ciemat.es de nuevo con el comando `yum`.

```
[root@router-mongos ~]# yum install ca-certificates
[root@router-mongos ~]# update-ca-trust enable # Permitimos la configuración CA dinámica
[root@router-mongos vi /etc/pki/ca-trust/source/anchors/

cp http://pki.irisgrid.es/ca/cert/9dd23746.crt -O /etc/pki/ca-trust/source/anchors/

[root@router-mongos ~]# update-ca-trust extract
```

Para cargar las variables de entorno utilizaremos:

```
[root@router-mongos ~]# source .creds/openrc-mbermejo
```

Procedemos a hacer unas cuantas pruebas para saber qué comandos tenemos que utilizar al comenzar a elaborar el script. [16]

```
[root@router-mongos ~]# nova image-list
```

ID	Name	Status	Server
31ac5c96-bd1e-4ca1-a82b-f74ba5586724	Centos 6.5 Minimal + Ssh	ACTIVE	5c35398b-33cf-4b6d-91ce-ba3b674fd334
9e496ee0-515a-4d66-b68e-fe9a349ddc3a	DB MongoDB Snapshots	ACTIVE	286dc9bb-daa4-4128-87c4-88d5c4b336b6
5460c884-7f79-4ea9-b2c9-54afecaac2c1	Maquina con Mongo	ACTIVE	e8c7e02e-5ad2-4ca5-9e52-f95a5352adeb

```
[root@router-mongos ~]# nova flavor-list
```

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
47f6e9a2-1bf4-4cdd-a686-d2de2f2d1356	ml-common	2048	20	0	2048	2	1.0	True
669614de-71eb-4c6a-ae0c-32a181a74b6f	ml-tiny	512	10	0		1	1.0	True
b33bf1fd-bf02-437d-99b1-adc4316b9d79	ml-small	2048	20	0		1	1.0	True
d80bfcf3-998c-44c1-9eed-2cf0ff2e270e	ml-large	8192	40	0		4	1.0	True
eb230fcc-6cfe-4172-9b8c-1628846382cb	ml-medium	4096	40	0		2	1.0	True
f47e2689-03c7-47fe-bbd2-d8546e00e2ca	ml-xlarge	8192	40	0		12	1.0	True

Podemos filtrar por la opción que siempre nos va a interesar, que será la que establezcamos para el lanzamiento de las máquinas (en nuestro caso, imagen: Maquina con Mongo; sabor: small).

Como ejemplo, vamos a crear un pequeño script que nos familiarizará con la nomenclatura `nova` y `shell script.`

A tener en cuenta: hemos creado anteriormente un par de claves que llamamos clave-maria.pem, el cual hemos descargado y luego subido al servidor. También vamos a lanzar la instancia cargando un archivo como si insertásemos en el Dashboard un script post-install, que nos servirá para asignarle un usuario y password a nuestra instancia, además de configurar sshd para que acepte contraseñas. Este archivo será pwd.file, también guardado en el servidor.

```
[root@router-mongos ~]# vi lanza_instancia.sh
 imagen= `nova image-list | grep Maquina | awk -f print2.awk`
#print2.awk guardado en el servidor es: {print $2} para evitar la nomenclatura de awk'{}'
#ya que da error.
 sabor= `nova flavor-list | grep small | awk -f print2.awk`

 nova boot --image "$imagen" --flavor "$sabor" --user-data pwd.file --
 key_name clave-maria.pem $instancia
#pwd.file contiene un scrip post-install para habilitar el ssh
#clave-maria.pem es el fichero generado para una clave pública para acceder mediante ssh -
i sin necesidad de ingresar contraseña.
```

Anexo.VI PSEUDOCODIGO

```
mientras 1=1
hacer
        storage = instruccion mongo que saca el storage total de las BDs
        replicaset = comando nova para listar el nºservidores que comienzan
        por mongodb-r y después lo dividimos entre 3
        valor = replicaset*valor teórico estimado en Mb.

        si $storage > $valor
        entonces
                Crear 2 archivos en directorios temporales para guardar datos y
                utilizarlos durante la creación de máquinas.

escribir "El Cluster está lleno, se procederá a ampliar la base de datos"
                #funciones
                CREARMAQUINAS ();
                ADDMVSH ();
        fin
hecho
```

```
funcion CREARMAQUINA () {
        Comando source para que estén disponibles los recursos nova
        imagen= "instrucción en mongo con grep y awk"
        flavor= "instrucción en mongo con grep y awk"

        #mongodb-rXy es la manera que tendremos de llamar a nuestras máquinas
        X=$replicaset+1
        for y en la secuencia de 0 a 2;
        hacer
                instancia = mongodb-r$X$y
escribir “Se crea la maquina $instancia”
                comando nova para creación de máquinas con $imagen y $flavor
                ip = comando nova show para sacar la ip de $instancia > fichero
                puerto= es ++1 de una variable global > archivoHosts
                enviar “$IP $puerto”/“$IP $instancia” a los archivos temporales
                (scp) Se enviará el /etc/hosts y el /etc/mongod.conf de la
maquina mongos a las nuevas máquinas para partir de configuraciones similares

        Hecho

Escribir “Terminado el proceso de creación de máquinas que formarán el RS”

LANZAR_ETC();

Hecho
```

```
función LANZAR_eETC() {


      arrayIP = comando nova para sacar todas las ip de los servidores
levantados
      arraySize = Size del arrayIP

escribir “Se procede a actualizar todos los /etc/host de todas las máquinas
con las nuevas IP Puerto de las maquinas creadas”

      for ((todas la sip del array));
      hacer

            (cat) conectar ssh $ip_maquina y actualizar fichero >> en
      directorio /etc/hosts

      done
}


función INICIARRS() {

      array_ipinstancia = lo sacamos del fichero temporal creado
      array_puertoinstancia = lo sacamos del fichero temporal creado

ip_primario = array[0]
puerto_primario = array[0]
ip_miembro1 = array[1]
puerto_miembro1 = array[1]
ip_miembro2 = array[2]
puerto_miembro2 = array[2]

escribir “ Se lanza la instancia mongo a las 3 nuevas maquinas”

      for ((cada ip de instancia del array));
      hacer
            conectar ssh $ip_instancia y lanzar el comando "mongod --dbpath
      --logpath -- etc" #segun la ip puerto de la máquina que corresponda

      hecho
escribir “Se inicializa el PRIMARIO y se añaden los dos miembros del RS”
conectamos ssh $ip_primario y lanzamos .js con initiate()
Sustituimos el fichero .js que va a añadir los miembros al replica set
(comando sed)
conectamos ssh ip_primario y lanzamos .js con rs.add()


}
```

función ADD_SH() ;

#Podría hacerse en la misma función anterior ahorrándonos el paso previo de sacar toda la info del fichero temporal

Utilizamos el mismo array sacando todas las ip / puerto y guardándolas en variables

Modificamos el fichero .js que vamos a lanzar sustituyendo por las variables adecuadas (comando sed)

mongo y lanzamos .js con sh.add()

#No hace falta conectarse a ninguna maquina ya que el sharding se maneja desde el mongos, maquina desde la cual lanzamos el script

}

Anexo.VII SISTEMA DE AUTOESCALADO DINÁMICO DE MONGODB SOBRE OPENSTACK

A continuación se muestra el Sistema de autoescalado final resultado de todo el Proyecto. Lleva la documentación necesaria para que sea totalmente comprensible por el usuario. Al finalizar se anexan 3 script .js que se utilizarán en el desarrollo de dicho sistema.

```
#!/bin/bash

function CREARMV () {

source .creds/openrc-mbermejo

      imagen=`nova image-list | grep "Maquina con Mongo" | awk '{print $2}'`
      sabor=`nova flavor-list | grep small | awk '{print $2}'`

   #Variable X va a determinar el nº del R-S en todas las funciones, por eso que sea global.
      X="$N_REPLSET + 1"
   #El bucle for se ejecuta 3 veces una por cada miembro. La variable del bucle será también
   variable para determinar el nº de miembro

      for (( y=0; y<=2; y++ ));
      do
            instancia="mongodb-r$X$y"
            source .creds/openrc-mbermejo
            nova boot --image "$imagen" --flavor "$sabor" --user-data
pwd.file "$instancia"

      #en pwd.file tenemos un script post-install que generará una clave pública al iniciar la
      instancia para poder acceder a ella por ssh sin contraseña.

            sleep 1m
            IP=`nova show "$instancia" | awk '$0~/internal-net/ {print $5}'`
            P=`expr $P + 1`
            echo "$IP $P" >> $fileIPP
            echo "$IP $instancia" >> $fileHosts

echo "Creada la primera máquina llamada $instancia con IP:$IP y Puerto:$P"
```

```
      #A cada instancia creada se le copia el /etc/host y /etc/mongod.conf para partir de una
      situación identica a la hora de lanzar la función LANZAR_ETC
              scp -i mongo-key.pem  /etc/hosts root@"$IP":/etc/hosts
              scp -i mongo-key.pem  /etc/mongod.conf
root@"$IP":/etc/mongod.conf

       done


       LANZAR_ETC();
       INICIARRS();
}



function LANZAR_ETC() {

      #Esta funcion se prepara para los usuarios que en mongoDB quieran utilizar, en vez de la
      IP como nombre de las maquinas, el hostname.


       ipArray=($(nova list | awk '$0~/internal-net/ {print $12}' | awk
'BEGIN{FS="[=,]"}{print $2}'))
       arraySize="${#ipArray[@]}"

      #Lanzamos la actualización del fichero hosts con las 3 nuevas IP y Puerto de las
      instancias creadas a todas las máquinas
       echo "Se procede a actualizar todos los /etc/hosts de todas las
máquinas con las 3 nuevas "IPs localhost" para la visibilidad entre máquinas
del Cluster"
       for (( i=0; i<${arraySize}; i++ ));
       do
              echo "Lanzamos el /ets/hosts a la maquina $ip_maquina"
              ip_maquina="${ipArray[$i]}"
              cat $fileHosts | ssh -i mongo-key.pem root@"$ip_maquina" ">>
/etc/hosts"
echo "--------------------------------Hecho--------------------------------"


       done


}
```

```
function INICIARRS() {


ipArray=($(awk '{print $1}' $fileIPP))
puertoArray=($(awk '{print $2}' $fileIPP))
arraySize="${#ipArray[@]}"

ip_primario="${ipArray[0]}"
puerto_primario="${puertoArray[0]}"

ip_miembro1="${ipArray[1]}"
puerto_miembro1="${puertoArray[1]}"
ip_miembro2="${ipArray[2]}"
puerto_miembro2="${puertoArray[2]}"

      #Vamos a ejecutar la instancia mongod en cada uno de los miembros del Replica Set. Que son
      los tres "ip puerto" del fichero $fileIPP


echo "---------Ejecutamos la instancia mongod en las 3 nuevas máquinas------"
       for (( i=0; i<"${arraySize}"; i++ ));
       do
              ip_maquina="${ipArray[$i]}"
              puerto_maquina="${puertoArray[$i]}"

echo "********************mongod para $ip $puerto**********************"

              ssh -i mongo-key.pem root@$ip_maquina "mongod --port
\"$puerto_maquina\" --replSet rs\"$X\" --dbpath /var/lib/mongo --logpath
/var/log/mongodb/log.log --logappend --smallfiles --oplogSize 50 --fork"

              ssh -i mongo-key.pem root@$ip_maquina "mongo --port
$puerto_maquina test --eval \"printjson (rs.status())\""

       done


      #A continuación, una vez todos con la instancia corriendo en modo rs, vamos a iniciar el
      PRIMARIO y a añadir los secundarios


echo "-------------Iniciamos el primer miembro en Modo Primario------------"
       mongo $ip_primario:$puerto_primario/test initiate.js
echo "-------------Y añadimos los demás miembros al replica set------------"
       sed -i "s/miembro1ip/$ip_miembro1/g" addrs.js
       sed -i "s/miembro1puerto/$puerto_miembro1/g" addrs.js
       sed -i "s/miembro2ip/$ip_miembro2/g" addrs.js
       sed -i "s/miembro2puerto/$puerto_miembro2/g" addrs.js


       mongo $ip_primario:$puerto_primario/test addrs.js
}
```

```
function ADDMVSH () {


ipArray=($(awk '{print $1}' $fileIPP))
puertoArray=($(awk '{print $2}' $fileIPP))
arraySize="${#ipArray[@]}"

ip_primario="${ipArray[0]}"
puerto_primario="${puertoArray[0]}"

ip_miembro1="${ipArray[1]}"
puerto_miembro1="${puertoArray[1]}"
ip_miembro2="${ipArray[2]}"
puerto_miembro2="${puertoArray[2]}"

echo "--------------Añadimos el Replica Set al Cluster Sharding-------------"
        sed -i "s/nºRS/$X/g" addsh.js
        sed -i "s/primario-ip/$ip_primario/g" addsh.js
        sed -i "s/primario-puerto/$puerto_primario/g" addsh.js
        sed -i "s/miembro1ip/$ip_miembro1/g" addsh.js
        sed -i "s/miembro1puerto/$puerto_miembro1/g" addsh.js
        sed -i "s/miembro2ip/$ip_miembro2/g" addsh.js
        sed -i "s/miembro2puerto/$puerto_miembro2/g" addsh.js

      mongo test addsh.js

      mongo test --eval \"printjson (sh.status())\"

}



# ------------------------CUERPO DEL SCRIPT--------------------------

while [true]

do

      storage=`mongo --eval " printjson(db.adminCommand('listDatabases'))" |
grep "sizeOnDisk" | awk -f awk1.awk`
      N_REPLSET=`nova list | grep db-r | awk '{x++;}END{print x/3}'`
      max_storage=`expr $N_REPLSET \* 512`    #Numero teórico que dependerá
del nº RS por un storage que decidamos


      if [$storage -gt $max_storage]
            then
echo "El Cluster está lleno, se procederá a ampliar la Base de Datos"
      cadena="search ceta-ciemat.es"
            if grep -Fxq "$cadena" /etc/resolv.conf
            then continue;
            else
            echo $cadena >> /etc/resolv.conf
      fi
```

```
            P=27005  #Actualmente tenemos una infraestructura con 2 Replica Set – 3
máquinas cada uno desde el puerto 27000 al 27005

            fileHosts="/tmp/hosts_update.txt"
            fileIPP="/tmp/IP_Puerto.txt"

      #funciones
            CREARMV();
echo "Replica Set creado y configurado con su PRIMARIO y dos secundarios,
Ahora lo añadiremos al Cluster Sharding"
            ADDMVSH();

            rm $fileHosts
            rm $fileIPP
      fi
done
```

Javascript:[6]

Initiate.js

```
function chequeo() {
        var res = null;
        printjson(res);
        if( !res || !res.ok ) {
              throw "No se ha cargado correctamente la configuración de
PRIMARIO";
        }
}
        print(db.isMaster().me);
        rs.initiate();
        sleep(100);
        chequeo();
```

addrs.js:

```
function chequeo() {
        var res = null;
        printjson(res);
        if( res || !res.ok ) {
        throw "No se ha cargado correctamente la configuración";
        }
}

function espera() {
        while( 1 ) {
              var res = rs.status();
              if( !res.ok )
                    continue;
              var x = false;
              for (var i = 0; i < res.members.length; i++ ) {
                    var state = res.members[i].state;
                    if( state != 1 && state != 2 )
                          x = true;
              }
              if( !x )

                    break;
        }
}
        rs.add(miembro1ip:miembro2puerto)
              sleep(100);
              chequeo();
        rs.add(miembro2ip:miembro2puerto)
              sleep(100);
              chequeo();
        espera();
```

addsh.js

```
function chequeo() {
        var res = null;
        printjson(res);
        if( !res || !res.ok ) {
        throw "No se ha cargado correctamente la configuración de
sharding";
        }
}
        sh.addShard("rs nºRS/primario-ip:primario-
puerto,miembro1ip:miembro1puerto,miembro2ip:miembro2puerto);
        chequeo();
```