#### Instituto Tecnológico de Costa Rica

**Escuela de Ingeniería en Computación**

**Bases de Datos Avanzadas**

**Proyecto#1:**

*“Se le pide diseñar e implementar sistema de reservación de tiquetes o boletos aéreos.  
La compañía aérea requiere que el sistema opere en un ambiente distribuido, el cual permitirá a los pasajeros la reservación de vuelos y la selección de asientos. La empresa tiene varias filiales. Los clientes podrán reservar un asiento para un vuelo, eligiendo si es en primera clase o en clase económica. Los(as) niños(as) hasta los 10 años tendrán un descuento del 10% en su boleto. Los clientes pueden cancelar o modificar sus reservaciones.  
Se requiere una alta disponibilidad del sistema. Los usuarios directos serán los empleados de la compañía que laboran en las diferentes filiales. Cada una de las cuales tendrá terminales enlazadas al sistema de reservación central”.*

**Desarrollado por:**

Leonardo Madrigal Valverde

Sebastián Segura Rosés

Christopher Vasquez Vargas

**Generalidades**:

**Database management system**

En este proyecto la herramienta que se utilizó para el manejo de la base de datos distribuida fue MongoDB, específicamente la versión 2.6.0

Se decidió utilizar MongoDB aprovechando sus mecanismos de distribución, además de su gran auge en desarrollo de aplicaciones. Esta herramienta se puede decir que es un sistema de base de datos no relacional, sencilla, escalable y orientada a los documentos.

El paquete MongoDB viene con un conjunto práctico de características, incluyendo:

**Sharding**

Tomando en cuenta que la fragmentación que se utilizó en el proyecto es de tipo horizontal, MongoDB va a cumplir un papel importante con su característica Sharding que facilita el crecimiento horizontal y la repartición de grupos de documentos (chunks) de las colecciones a lo largo de una serie de nodos, distribuidos de forma uniforme.

De esta manera se puede analizar Sharding como un método que utiliza en este caso MongoDB para almacenar datos a través de varias máquinas, para el manejo de conjuntos grandes de datos y operaciones de alto rendimiento sobre estos datos (Si se valora este requerimiento de un gran rendimiento sobre el manejo de grandes cantidades de datos, es posible que cuestione la capacidad de un servidor.) En fin , es un buen método que permite el manejo adecuado del tipo de distribución que se adoptó en el proyecto, asignando los datos a través de servidores o fragmentos, en donde cada fragmento es una base de datos independiente, y el conjunto de estos fragmentos conformaran una única base de datos lógica.

**Aplicación**

**Lenguaje C#**

En este caso el lenguaje de programación que se ha utilizado para implementar la aplicación es C#, un lenguaje de programación que se ha diseñado para compilar diversas aplicaciones que se ejecutan en .NET Framework. Se puede analizar a C# como un lenguaje simple, eficaz, con seguridad de tipos y orientado a objetos. Un punto importante en este lenguaje es que sus versiones más recientes poseen innovaciones tales que permiten desarrollar aplicaciones tan elegantes y expresivas como lo puede ser una aplicación desarrollada en lenguaje C.

**IDE Visual Studio**

Tomando en cuenta una colección de herramientas y servicios que le permitirá crear la aplicación se escogió el entorno de desarrollo Visual Studio que será un IDE de gran ayuda con su editor de código con buenas características, compilador, plantillas de proyecto, diseñadores, asistentes de código, depurador y también se puede decir que es relativamente fácil de usar.

**Entorno de ejecución .NET Framework**

Por otra parte la biblioteca de clases de .NET Framework ofrece acceso a numerosos servicios de sistema operativo y a otras clases útiles y adecuadamente diseñadas que aceleran el ciclo de desarrollo de manera significativa, por lo que se utilizó .NET Framework 4.5 el cual incorpora

* Common Language Runtime, que proporciona administración de la memoria y otros servicios del sistema.
* Una biblioteca de clases completa, que permite aprovechar código probado y reutilizable al que se puede llamar desde la aplicación.

**E-R global y locales**

Para interpretar de manera correcta los términos y así también la forma con que Mongo maneja esos conceptos respecto a SQL, se tomó del website de MongoDB el siguiente cuadro comparativo:

| SQL Terms/Concepts | MongoDB Terms/Concepts |
| --- | --- |
| database | [database](http://docs.mongodb.org/manual/reference/glossary/#term-database) |
| table | [collection](http://docs.mongodb.org/manual/reference/glossary/#term-collection) |
| row | [Document](http://docs.mongodb.org/manual/reference/glossary/#term-document),  [BSON](http://docs.mongodb.org/manual/reference/glossary/#term-bson) document |
| column | [field](http://docs.mongodb.org/manual/reference/glossary/#term-field) |
| index | [index](http://docs.mongodb.org/manual/reference/glossary/#term-index) |
| table joins | embedded documents and linking |
| primary key  Specify any unique column or column combination as primary key. | [primary key](http://docs.mongodb.org/manual/reference/glossary/#term-primary-key)  In MongoDB, the primary key is automatically set to the[\_id](http://docs.mongodb.org/manual/reference/glossary/#term-id) field. |
| aggregation (e.g. group by) | aggregation pipeline |

Ahora de esta manera se puede analizar MongoDB tomando en cuenta que es un modelo no relacional, el cual maneja de manera distinta el modelo de sus datos, si lo vemos desde el punto de vista de un modelo de entidad-relación hacia el modelo que se utilizó en MongoDB, lo primero que se debe hacer es pensar sobre las operaciones CRUD básicas en MongoDB: ya sean insert, find, update y  remove.

**Diseño global**

Vuelos =

{

\_id: // identificador del vuelo

hora: // hora del vuelo

fecha:{ // fecha compuesta del vuelo

dia:

mes:

anio:

}

asientosDisponibles: // asientos que no han sido vendidos

estadoVuelo: // muestra el estado en que esta el vuelo en un determinado momento

paisSalida: // país de origen del vuelo

paisDestino: // país al que arribará el vuelo

}Tiquetes =

{

\_id: // identificador del tiquete

infoCliente: { // información del cliente que comprará el tiquete

idCliente:

pasaporte:

edad:

}

infoVuelo: { // información compuesta del vuelo que tomará el cliente

idVuelo:

hora:

fecha: {

dia:

mes:

anio:

}

}

asientos: // los asientos que se asignará a la compra del tiquete

precio: // precio del tiquete

estadoPago: // si el tiquete se ha cancelado ya o no

paisSalida: // país de origen del vuelo

paisDestino: // país al que arribará el vuelo

}

Asientos =

{

\_id: // identificador del asiento

estadoReservacion: // Estado de la reserva: P=’pagada’, S=’sin pagar’

tipoAsiento: // tipos P = ‘primera clase’, N = ‘negocios’, E= ‘económica’

precio: // precio del asiento escogido

}

AsientosPorVuelo =

{

idVuelo: // identificador del vuelo

asientos: [] // asiento seleccionados

}Clientes =

{

\_id: // identificador del cliente

nombre: // nombre del cliente

apellido1: // primer apellido del cliente

apellido2: //segundo apellido del cliente

edad: // edad del cliente

pasaporte: // N° de pasaporte del cliente

telefono: // N° Teléfono del cliente

genero:

paisRegistro: // país donde se registró el cliente

}Reservaciones:

{

\_id: // identificador de la reservación

datosCliente: { // datos compuestos del cliente que hace la reservación

idCliente:

pasaporte:

edad:

}

fecha:{ // fecha compuesta de la reservación

dia:

mes:

anio:

}

hora: // hora de la reservación

vuelo: // vuelo de la reservación

asientos: // asientos de la reservación

paisRegistro: // país de registro de la reservación }

ColaReservaciones =

{

idReservacion: // identificador de la reservación

paisRegistro: // país donde se realizó la reservación

hora: // hora de la reservación

fecha:{ // fecha compuesta en que se realizó la reservación

dia:

mes:

anio:

}

}

Empleados =

{

\_id: nombre: // identificador del empleado

apellido: // apellido del empleado

usuario: // nombre de usuario del empleado

password: // clave de acceso del empleado

pais: // país donde labora el empleado

}

**Diseño local**

Vuelos =

{

\_id: // identificador del vuelo

hora: // hora del vuelo

fecha:{ // fecha compuesta del vuelo

dia:

mes:

anio:

}

asientosDisponibles: // asientos que no han sido vendidos

estadoVuelo: // muestra el estado en que esta el vuelo en un determinado momento

paisSalida: // país de origen del vuelo

paisDestino: // país al que arribará el vuelo

}Tiquetes =

{

\_id: // identificador del tiquete

infoCliente: { // información del cliente que comprará el tiquete

idCliente:

pasaporte:

edad:

}

infoVuelo: { // información compuesta del vuelo que tomará el cliente

idVuelo:

hora:

fecha: {

dia:

mes:

anio:

}

}

Clientes =

{

\_id: // identificador del cliente

nombre: // nombre del cliente

apellido1: // primer apellido del cliente

apellido2: //segundo apellido del cliente

edad: // edad del cliente

pasaporte: // N° de pasaporte del cliente

telefono: // N° Teléfono del cliente

genero:

paisRegistro: // país donde se registró el cliente

}Reservaciones:

{

\_id: // identificador de la reservación

datosCliente: { // datos compuestos del cliente que hace la reservación

idCliente:

pasaporte:

edad:

}

fecha:{ // fecha compuesta de la reservación

dia:

mes:

anio:

}

hora: // hora de la reservación

vuelo: // vuelo de la reservación

asientos: // asientos de la reservación

paisRegistro: // país de registro de la reservación

}

**Diseño de transacciones**

Para el diseño de transacciones se utiliza una función de MongoDB llamada FindAndModify(). Esta función actualiza el registro de manera atómica, esto es una transacción.

**Diseño de la fragmentación**

El tipo de fragmentación que se utilizó fue la fragmentación horizontal, tomando en cuenta la existencia de 3 nodos (Costa Rica, Brasil y Panamá), se fragmentaron las colecciones de clientes, tiquetes, vuelos y reservaciones, distribuyendo los documentos en los ya mencionados nodos, siguiendo los siguientes criterios:

**Clientes:** el campo “paisRegistro” hace referencia a la filial del sistema en donde el usuario se registró. De esta manera se distribuyen en cada una de las 3 filiales.

* paisRegistro = “Costa Rica”
* paisRegistro = “Brasil”
* paisRegistro = “Panama”

**Reservaciones:** la fragmentación de esta colección se realizó de igual manera que la colección de clientes, puesto que se particiona según el país en donde se procesó la reservación.

**Vuelos:** el campo “paisSalida” hace referencia al país de donde parte el vuelo. Para efectos de este proyecto, se manejaron los vuelos de manera en que los aviones parten de alguno de los 3 países en donde se encuentran las sucursales y poseen algún otro país destino.

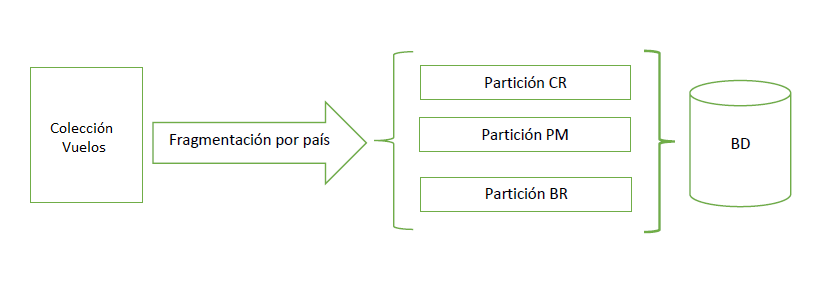
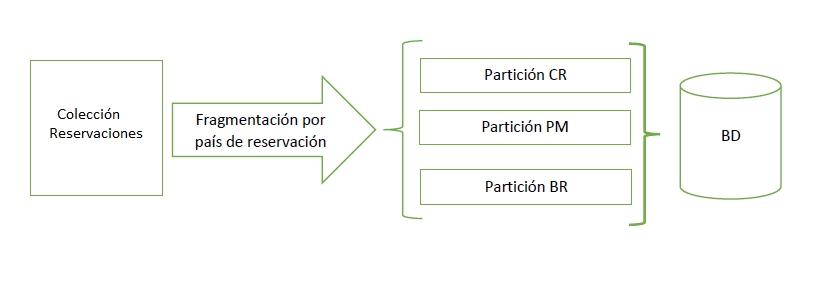
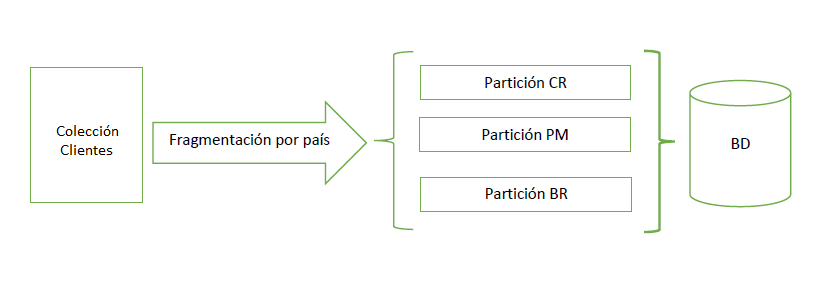
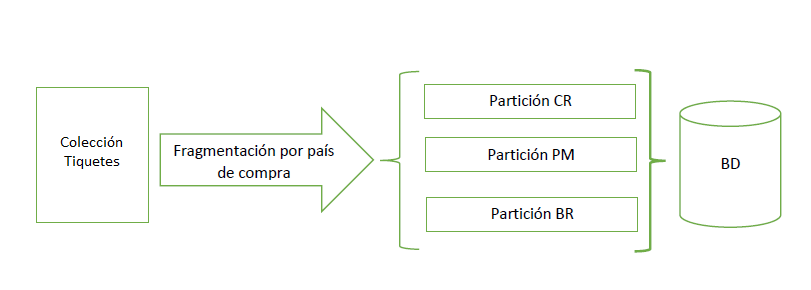
* paisSalida = “Costa Rica”
* paisSalida = “Brasil”
* paisSalida = “Panama”

**Tiquetes:** de igual forma, los criterios para su distribución son de acuerdo al país de donde partirá el avión, por lo tanto, los predicados utilizados fueron los mismos para las colecciones Tiquetes y Vuelos.

**Diseño de la asignación**

Al seleccionar como método de distribución la fragmentación horizontal, utilizando como criterio solamente los países en donde se realizaron las transacciones de compra y reservación de tiquetes; además, al asumir que los 3 nodos se encuentran en los países de Costa Rica, Brasil y Panamá, la asignación corresponde a (valga la redundancia) asignar a cada nodo, los documentos que posean como criterio de partición el nombre del país en donde ocurrieron los eventos, ya sea de registro o de la partida de un avión.

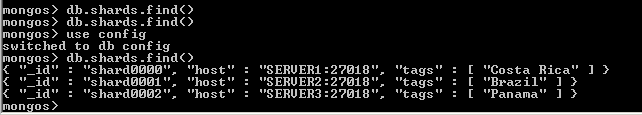
Así pues, se representa mediante los siguientes diagramas, la fragmentación y asignación utilizados para el proyecto:



Además, en la sección de Anexos se presenta una serie de instrucciones, en las cuales se explica la manera en la cual se utilizó la característica de Sharding presente en el SABD MongoDB para lograr la distribución y la asignación de la base de datos que se utilizó.

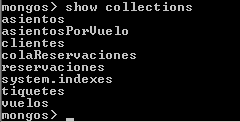
**Evidencias**

Antes que nada, se pone en evidencia la existencia de los “shards” o nodos que se utilizaron, con sus respectivos “tags” o nombres, mediante la siguiente imagen:



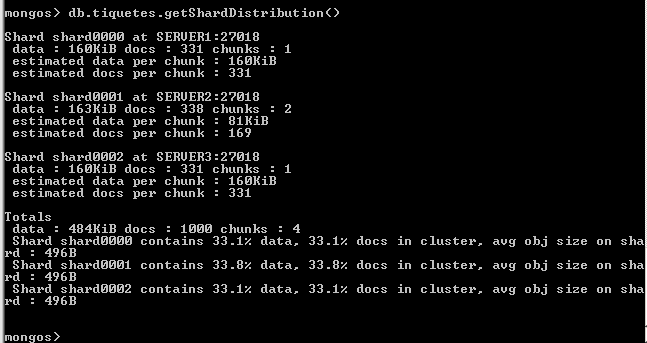
En donde “SERVER1”, “SERVER2” y “SERVER3” son miembros de un dominio creado en Windows Server 2008 R2, implementados en máquinas virtuales utilizando el software Oracle Virtual Box.

Ahora, la siguiente imagen representa la estructura previamente explicada de la base de datos, luego de ejecutar un script de llenado realizado en javascript:



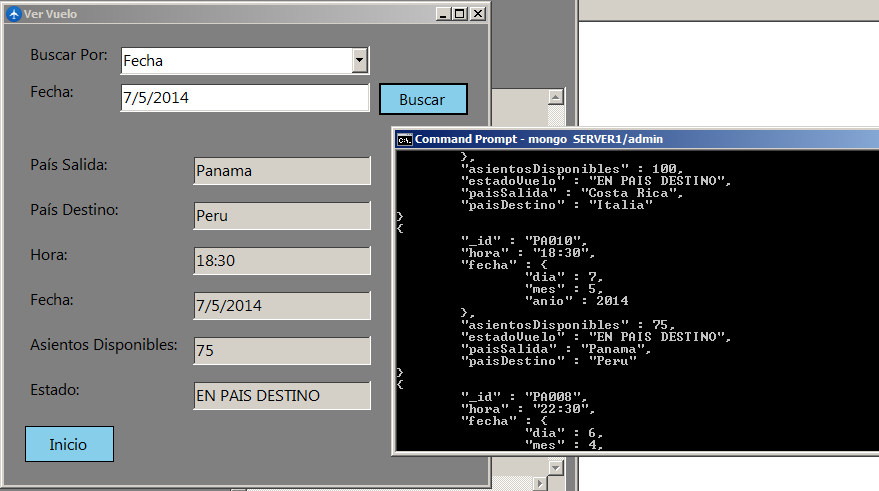
Y las siguientes imágenes hacen referencia al estado de las colecciones que fueron almacenados en la base de datos, primero con la colección “asientos”, la cual es una colección que no fue fragmentada y, después, con la colección “tiquetes”, la cual sí lo fue. Esto para demostrar que sí hubo una distribución.

SS3

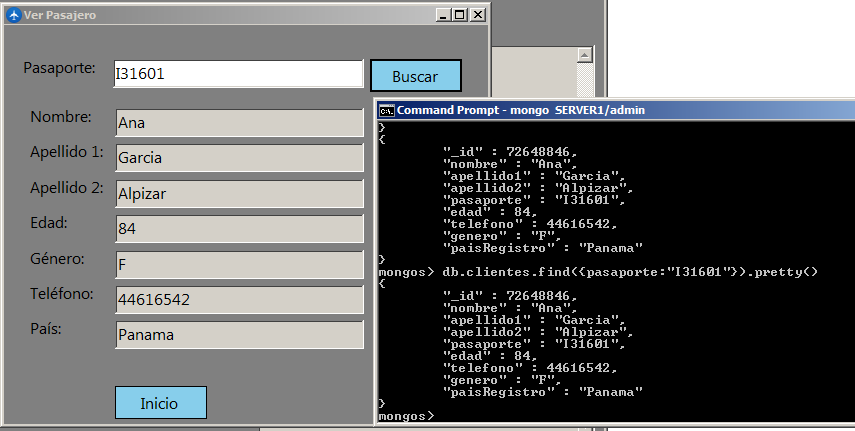


**Funcionamiento**

A continuación se presentan imágenes que muestran el resultado de consultas de algunas funciones del sistema:



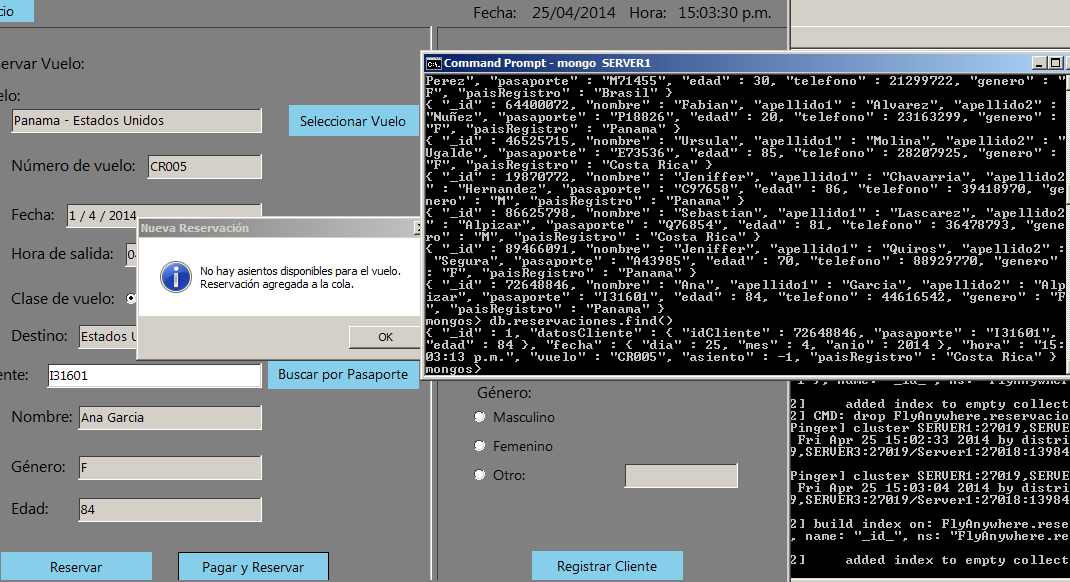
En esta imagen se muestra la consulta de un vuelo por medio de la fecha en que partirá y al lado de la ventana de la aplicación, se ve la consola de MongoDB que comprueba que la consulta está buena



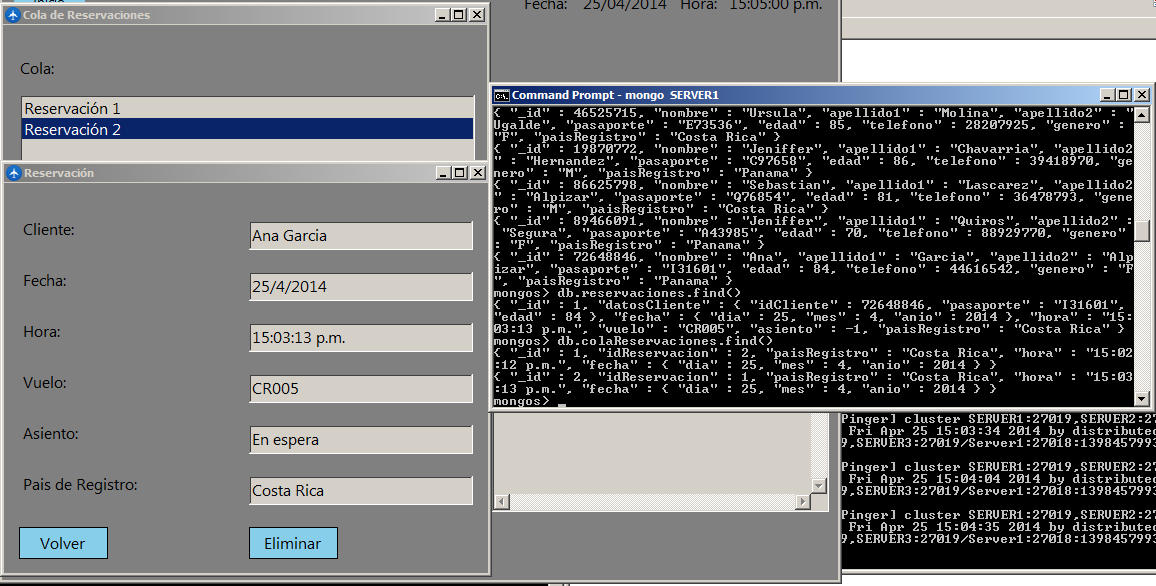
Esta es otra consulta, pero esta vez a la colección de pasajeros, utilizando como criterio el pasaporte, de nuevo, al lado de la ventana de la aplicación se muestra la consola de mongo que corrobora la funcionalidad.

Ahora, se mostrará el estado de la base de datos, en la colección de reservaciones, antes de realizar una reservación. Se puede observar que no hay ningún registro en ese momento:

Ahora se muestra que se realizó con éxito una reservación para un vuelo, sin embargo, no habían espacios válidos, por lo tanto la reservación también fue añadida a la cola de reservaciones (lista de espera):



Y por último, se puede ver el estado de la colección “colaReservaciones”



**Referencias**

* Página web oficial de MongoDB: <https://www.mongodb.org/>
* Documentación de MongoDB: <http://docs.mongodb.org/manual/>
* Documentación de C# MongoDB Driver: <http://docs.mongodb.org/ecosystem/tutorial/use-csharp-driver/>
* **Anexos**

A continuación se presenta un corto tutorial para lograr una distribución de una base de datos en MongoDB (versión 2.6.0), en la cual se utilizan las características de Sharding y de Tag Aware Sharding para la distribución y asignación.

1. Para iniciar el servidor de un shard (hacer esto en n máquinas para tener n shards):

mongod --shardsvr

1. Para iniciar los config server (hacer en por lo menos 3 máquinas):

mongod --configsvr –dbpath /data/configdb --port 27019

1. Se inicia el servicio de mongo, se debe iniciar en las máquinas en donde se ejecutarán las aplicaciones:

mongos --configdb SERVER1:27019,SERVER2:27019,SERVER3:27019

1. Para añadir Tags que referencian a shards específicos:

use admin

sh.addShardTag("shard000n", "<nombreDelTag>")

\*Donde n depende de la cantidad de shards que se hayan creado.

\*\*Se tiene que hacer para cada uno de los shards que se desean asociar con un rango específico para las shardKeys.

1. Para permitir el sharding en una base de datos:

sh.enableSharding("<nombreDB>")

1. Para particionar una colección donde shardKey es el nombre del atributo en donde se quiere particionar:

sh.shardCollection("[nombreDB].[nombreCollection]", { shardKey : 1})

\*Se debe hacer para cada colección que se desea particionar.

1. Para añadir rangos para los cuales se repartirán los registros en los distintos shards:

sh.addTagRange("[nombreDB].[nombreCollection]", {shardKey: rangoMínimo}, {shardKey: rangoMáximo}, "<shardTag>")

\*RangoMínimo y rangoMáximo SON inclusivos.

\*\*Se agregan los rangos que se deseen.

\*\*\*Los sharTags ya deben de estar definidos

1. Para ver la distribución de los datos en los distintos shards:

use <nombreDB>

db.<collection>.getShardDistribution()