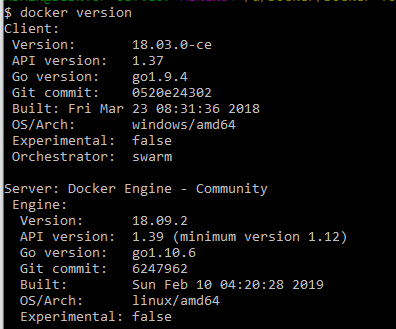
**Docker y MongoDb**

**Algunos comandos de docker:**

* **docker info**: Ver información sobre Docker en todo el sistema.
* **docker version:** Indica versión tanto del cliente como del servidor.



* **docker --help:** Visualiza la lista completa de los subcomandos disponibles.
* **docker pull *nombre\_imagen:*** descarga imagen.
* **docker images:** ver imágenes existentes.
* **docker images --help:** ver la lista completa de los subcomandos para las imágenes.
* **docker images -a:** ver todas las imágenes.
* **docker** **images -u:** ver solo id de imágenes.
* **docker rmi $(docker images -q):** borrar todas las imágenes.
* **docker run *nombre\_imagen:*** crear imagen a partir de container en hub.
* **docker run *nombre\_imagen* ls:** crea container con imagen ubuntu y con **run** a parte de crear container permite ejecutar comandos como **ls** de ubuntu para ver carpeta de directorios.
* **docker ps:** ver containers corriendo.
* **docker ps -a:** ver containers tanto activos(corriendo) o inactivos.
* **docker rm *num\_id\_container (o primeros 2 o 3 dígitos del id\_container)***: borrar container.
* **docker rm *num\_id\_container1 num\_id\_container2***: borrar mas de un container.
* **docker run -i -t *nombre\_imagen* bash:** crea container que se ejecute en terminal dentro de la terminal de la imagen (bash). el comando **-i** es para indicar que es interactivo, el **-t** indica terminal o pseudo terminal y bash es para ingresar a la terminal del container ubuntu en este caso.
* **docker run -i -t *nombre\_imagen1* bash; docker commit container\_id *nombrenuevaimagen*:** con **commit** se crea una nueva imagen a partir del contenedor creado (***nombre\_imagen1***), luego se crea un nuevo contenedor a partir de esa imagen creada (imagen ***nombrenuevaimagen***).
* **docker run -d -it *nombre\_imagen1*:** crea un contenedor interactivo (**-it**) y queda corriendo por el comando **-d**.
* **docker run -it nombre\_imagen:versión:** crea un contenedor interactivo con una imagen y versión puntual (ej: docker run -it ubuntu:14.04 /bin/bash).
* **docker rmi *imagen\_id*:** borrar imagen.
* **ctrl + p + q:** para salir del contenedor pero dejarlo corriendo.
* **docker** **start *id\_container* *(o primeros 2 o 3 dígitos del id\_container)*:** para activar o correr contenedor.
* **docker** **stop *id\_container* *(o primeros 2 o 3 dígitos del id\_container)*:** para parar contenedor.
* **docker attach *id\_container*:** conectarse a un contenedor corriendo en bash (terminal).
* **docker network ls:** ver redes.
* **docker network create nombre\_red:** crear red.
* **docker network inspect *bridge* (o *host*):** ver características de red.
* **docker network rm *nombre\_red***: borrar una red.
* **docker rm $(docker ps -a -q *-f*):** borra todos los contenedores, con **-f** se fuerza el borrado.
* **docker logs *container\_di*:** ver logs en la consola.
* **-p <puerto\_host>:<puerto\_contenedor>:** acceder desde fuera del contenedor, se debe mapear o hacer mapeo de puertos que queremos publicar con la bandera **-p (-publish).**
* **-p <ip>:<puerto\_host>:<puerto\_contenedor>:** acceder desde fuera del contenedor, se debe mapear o hacer mapeo de puertos que queremos publicar con la bandera **-p (-publish)** con <ip> especifica la interfaz de red a utilizar (ej: docker run -d -p 8080:8080 tomcat)**.**
* **docker inspect id\_container:** revisar configuaraciones de la red, imprime configuración en formato Json.
* **docker inspect --format '{{ .NetworkSettings.IPAddress }}' <container\_id>:** para mostrar solo la dirección ip asignada al contenedor.
* **docker inspect -f '{{range.NetworkSettings.Networks}} {{.IPAddress }} {{ end }}' <container\_id>:** ver la ip asignada.
* **bridge:** standar que usarán los contenedores**, none:** indica que un contenedor not iene asignada una red**, container:** red única con otro contenedor especificado**, host:** el contenedor usará el mismo IP del servidor real que tengamos. Red de anfitrión(local) sin firewall**.**
* **EXPOSE:** exponer un puerto por defecto.
* **docker run -d -P *imagen\_nombre*:** puerto de forma automática.
* **docker port <container\_id>:** ver puerto.
* **docker network create --driver bridge *mi\_red*:** crear nueva red se podrá usar para aislar ciertos contenedores.
* **docker run -id -P --name *contenedor\_nombre* --network *mi\_red imagen\_uno*:** si arrancamos 2 contenedores indicando que usen esa red con “-network xxx” ambos obtendrán un IP de ese rango (172.19.0.2 y 3) y se verán entre sí.
* **docker network connect --link container1:red2 nombre\_red\_virtual container2**

**docker network connect --link container2:red2 nombre\_red\_virtual container1:** ambos contenedores a otra red creada manualmente (mi\_net)**.**

* **docker network disconnect *nombre\_red container\_nombre*:** desconectar contenedor de la red.
* **docker network prune:** elimina todas las redes que no estén siendo usadas por contenedores.
* **mongod --port 27666 --dbpath data --replSet rep1:** Co**mo ya sabéis *mongod*es el nombre del proceso (y del ejecutable) que hace de servidor de MongoDB. Con el primer parámetro indicamos el puerto por el que nuestro servidor va a escuchar peticiones. Por defecto MongoDB utiliza el 27017, pero en este caso lo he cambiado por algo más sencillo de recordar. Tened en cuenta que vamos a ejecutar tres instancias de MongoDB en la misma máquina, y que cada una de ellas deberá escuchar por un puerto distinto.**

**Con el parámetro *dbpath*estamos indicando dónde están nuestros datos, que en este caso están en un directorio llamado *data*, que está al mismo nivel que *mongod*. Para finalizar el parámetro más importante,*replSet*, que indica que estamos metiendo nuestro servidor en un conjunto de réplicas que se llama *rep1*.**

**Al ejecutar el comando veréis que el servidor arranca. Si dejáis un rato la ventana funcionando, se puede ver que no para de mostrarse el siguiente mensaje**

**[rsStart] replSet can't get local.system.replset config from self or any seed (EMPTYCONFIG)**

**Esto es porque aunque hemos metido el servidor en un conjunto de réplicas, no lo hemos inicializado.**

* **docker exec -it *nombre\_contenedor* *imagen\_nombre*:** ingresa al shell de mongo interactivo cuando el contenedor ya está corriendo (**docker exec -it mongo1mongo**).
* **docker**
* **Para copiar archivos / carpetas entre un contenedor y el sistema de archivos local, escriba el comando:**

**docker cp {SOURCE\_FILE} {DESTINATION\_CONTAINER\_ID}:/{DESTINATION\_PATH}**

**Ejemplo:**

**docker cp /home/foo container-id:/home/dir**

* **Importar los datos a nuestro servidor de MongoDB:**

**mongoimport --db ciudadesdb --collection ciudades --drop --file zips.json --port 8080**

Con el parámetro “–db” le indicamos que queremos guardar los datos en la siguiente base de datos. En este caso “ciudadesdb”.

Con el parámetro “–collection” le indicamos la colección, es decir, la tabla donde queremos guardar los datos. En este caso “ciudades”.

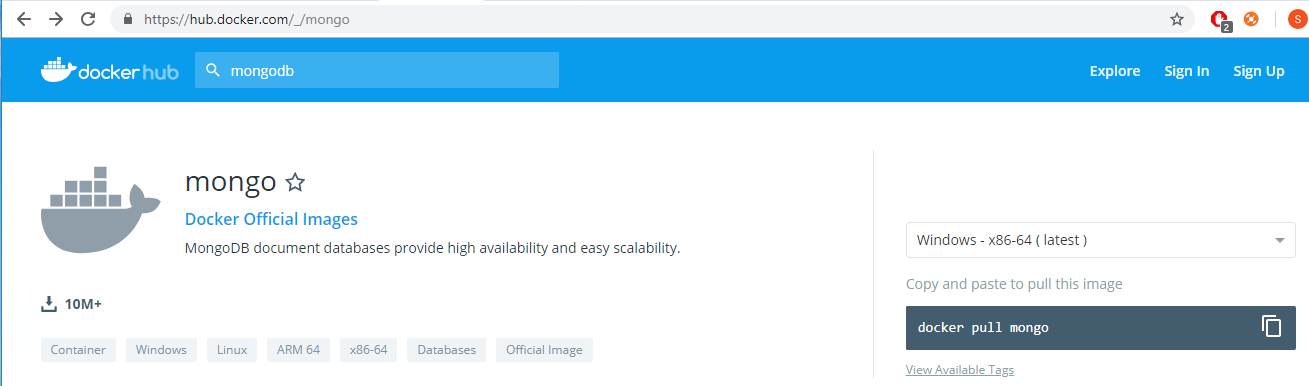
El parámetro “–drop” le indicar a mongoimport que descarte todos los datos que hubiese ya en la colección. Es decir, borraría todos los datos que tuviese la tabla antes de realizar la importación de los datos. Cuidado con este parámetro, si queremos añadir los nuevos datos a los ya existentes, no debemos incluir este parámetro.

El parámetro “–file” es donde le indicamos a mongoimport donde está el fichero en formato json con los datos que deseamos importar.

Para indicarle a mongoimport en qué dirección está nuestro servidor usaremos el paramento “–host” y le indicaremos la dirección donde se encuentra nuestro servidor.

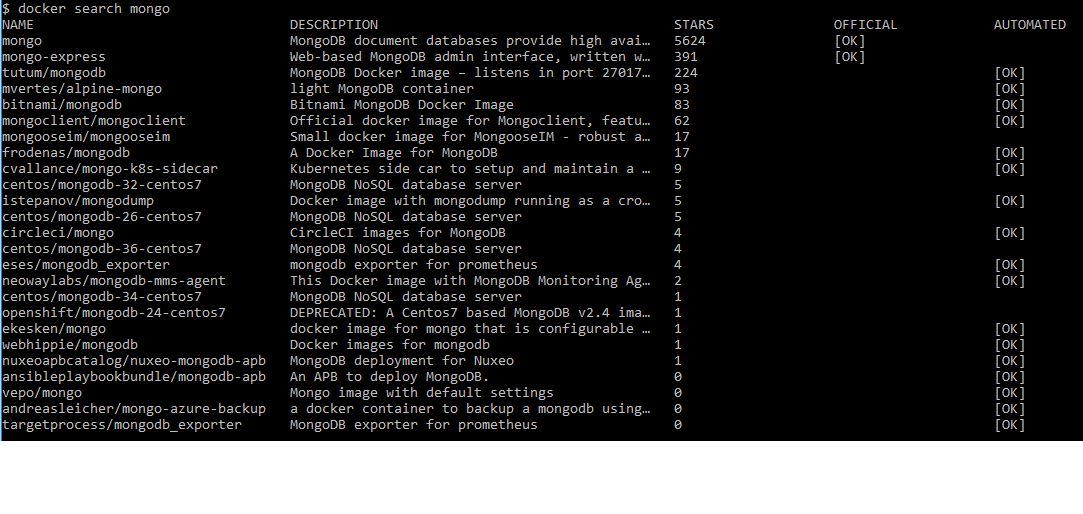
Para indicarle el puerto por que cual está a la escucha nuestro MongoDB utilizaremos el paramento “–port” y el puerto que hemos configurado.

Para iniciar con Mongo en Docker se necesitará primero descargar la imagen de mongo por medio de la página para buscar las imágenes de Docker: <https://hub.docker.com> buscando la imagen correspondiente:



En la página de imágenes de Docker se indicará el comando para descargar la imagen de mongo en Docker.

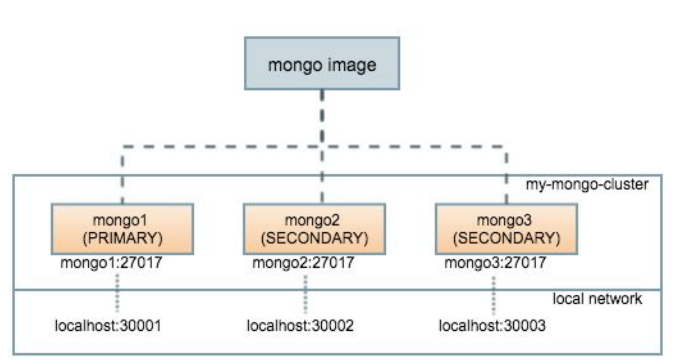
Otra forma de buscar las imágenes de Docker es por consola por medio del comando: docker search nombre\_imagen con el cual se mostrarán los resultados encontrados con los nombres de la imagen.



* **Instalar imagen de mongo:**

Se instala la imagen en docker con el comando: docker pull mongo.

* **Docker - MongoDB: Replica set:**

****

**Ejemplo:** https://www.sohamkamani.com/blog/2016/06/30/docker-mongo-replica-set/

Vamos a tener 3 contenedores de la imagen de Mongo, todos dentro de su propia [red de contenedores docker](https://docs.docker.com/engine/userguide/networking/dockernetworks/) . Vamos a nombrar ellos mongo1, mongo2y mongo3. Estas serán las tres instancias mongo de nuestro conjunto de réplicas. También vamos a exponer cada uno de ellos a nuestra máquina local, de modo que podamos acceder a cualquiera de ellos usando la interfaz de shell de Mongo desde nuestra máquina local si es necesario (tendrá que instalar MongoDB en su propia máquina para hacer esto) . Cada uno de los tres contenedores mongo debe poder comunicarse con todos los demás contenedores de la red.

1. **Se crea una red:**

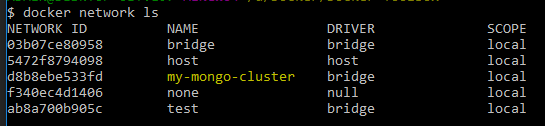
Vamos a crear un red virtual en docker sobre la cual se conectarán

nuestros nodos del replica set:

**$ docker network create my-mongo-cluster**



**$ docker network ls:** vemos las redes que hay.



1. **Se crean los contenedores:**

Vamos a crear contenedor llamado mongo1, usando la imagen de

docker que descargamos. Este será el nodo maestro.

**$ docker run -p 30001:27017 --name mongo1 --net my-mongo-cluster mongo mongod --replSet my-mongo-set**

* **docker run :** Iniciar un contenedor desde una imagen
* **-p 30001:27017 :** Expone el puerto 27017 en nuestro contenedor, como el puerto 30001 en el localhost
* **--name mongo1 :** nombre este contenedor "mongo1"
* **--net my-mongo-cluster :** Agregue este contenedor a la red "my-mongo-cluster".
* **mongo :** el nombre de la imagen que estamos usando para generar este contenedor
* **mongod --replSet my-mongo-set :** Ejecute mongod mientras agrega esta instancia mongod al conjunto de réplicas denominado "my-mongo-set"
* **mongod:** es el nombre del proceso (y del ejecutable) que hace de servidor de **MongoDB.**

Configura los otros 2 contenedores ejecutando:

**$ docker run -p 30002:27017 --name mongo2 --net my-mongo-cluster mongo mongod --replSet my-mongo-set**

**$ docker run -p 30003:27017 --name mongo3 --net my-mongo-cluster mongo mongod --replSet my-mongo-set**

1. **Configuración de la replicación:**

Ahora que tenemos todas nuestras instancias de mongo funcionando, vamos a convertirlas en un conjunto de réplicas.

Conéctate al mongo shell en cualquiera de los contenedores.

**$ docker exec -it mongo1 mongo**

Este comando abrirá el shell mongo en nuestro contenedor mongo1 en ejecución (pero también puede ejecutarlo desde el contenedor mongo2 o mongo3).

Dentro del shell mongo, primero creamos nuestra configuración:

MongoDB shell version: 2.6.7

Asigna un identificador para seguir los comandos del shell en mongo:

**> db = (new Mongo('localhost:27017')).getDB('test')**

**test**

**> config = {"\_id" : "my-mongo-set","members" : [{"\_id" : 0,"host" : "mongo1:27017"}, {"\_id" : 1,"host" : "mongo2:27017"}, {"\_id" : 2,"host" : "mongo3:27017"}]}**

La primera clave \_id en la configuración debe ser la misma que la marca **--replSet** que se estableció para nuestras instancias mongod, que es **my-mongo-set** en nuestro caso. Luego listamos todos los miembros que queremos en nuestro conjunto de réplicas. Desde que agregamos todas nuestras instancias mongo a nuestra red docker. Su nombre en cada contenedor de resolución a sus respectivas direcciones ip en la red **my-mongo-cluster**.

Finalmente comenzamos el conjunto de réplicas ejecutando:

**> rs.initiate(config)**

**{ "ok" : 1 }**

En nuestra shell de mongo. Si todo va bien, su mensaje debería cambiar a algo como esto:

**my-mongo-set:PRIMARY>**

Esto significa que el shell está asociado actualmente con la base de datos PRIMARY en nuestro my-mongo-set clúster.

Vamos a jugar un poco con nuestro nuevo conjunto de réplicas para asegurarnos de que funciona según lo previsto. *(Estoy omitiendo el my-mongo-set:PRIMARY>mensaje de legibilidad)*

Primero insertamos un documento en nuestra base de datos primaria:

**> db.mycollection.insert({name : 'sample'})**

WriteResult({ "nInserted" : 1 })

**> db.mycollection.find()**

{ "\_id" : ObjectId("57761827767433de37ff95ee"), "name" : "sample" }

Luego hacemos una nueva conexión a una de nuestras bases de datos secundarias (ubicadas en mongo2) y las comprobamos para ver si nuestro documento también se replica allí:

**$** **docker exec -it mongo2 mongo**

**> db2 = (new Mongo('mongo2:27017')).getDB('test')**

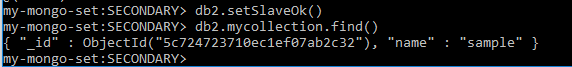
test

**> db2.setSlaveOk()**

**> db2.mycollection.find()**

{ "\_id" : ObjectId("5c724723710ec1ef07ab2c32"), "name" : "sample" }

Ejecutamos el comando **db2.setSlaveOk()** para que el shell sepa que estamos consultando intencionalmente una base de datos que no es nuestra principal (indicar que es un esclavo). Y parece que el mismo documento está presente en nuestra secundaria también.



Confirmamos la tercera:

**$** **docker exec -it mongo3 mongo**

**Se podría acceder a los comandos con db, ejemplo: db.setSlaveOk() pero al poner el identificados db3 ya se puede llamar así mismo.**

**> db3 = (new Mongo('mongo3:27017')).getDB('test')**

test

**> db3.setSlaveOk()**

**> db3.mycollection.find()**

{ "\_id" : ObjectId("5c724723710ec1ef07ab2c32"), "name" : "sample" }

## **Avanzando**

Como puede ver, con el poder de la ventana acoplable pudimos configurar y ejecutar una réplica de Mongo en aproximadamente 5 minutos. Aunque esta configuración es genial para experimentar y jugar con conjuntos de réplicas, hay algunas precauciones que deben tomarse antes de pasar a producción:

* Ninguna de las bases de datos tiene medidas de seguridad administrativa. Asegúrese de agregar usuarios y contraseñas al implementar esta solución en un servidor real.
* Mantener todos los contenedores en un solo servidor no es la mejor idea. Ejecute al menos un contenedor en un servidor diferente y acceda a él a través de su dirección IP externa y puerto (en nuestro caso, los puertos externos para los contenedores externos fueron 30001, 30002 y 30003 para mongo1, mongo2 y mongo3 respectivamente).
* En caso de que eliminemos uno de nuestros contenedores por error, los datos también desaparecerán. El uso de [volúmenes Docker](https://docs.docker.com/v1.10/engine/userguide/containers/dockervolumes/) y la configuración adecuada --dbpathcuando se ejecuta mongodevitarían que esto sucediera.

Finalmente, en lugar de ejecutar un montón de scripts de shell, puede que le resulte más conveniente automatizar todo este proceso mediante el uso de herramientas de automatización de múltiples contenedores, como la función [docker-compose](https://docs.docker.com/compose/) .

1. **Cargar colección:**

Ruta archivo colección:

<https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json>

Pasamos la colección de la máquina local al contenedor maestro (mongo1):

**$ docker cp primer-dataset.json mongo1:/primer-dataset.json**

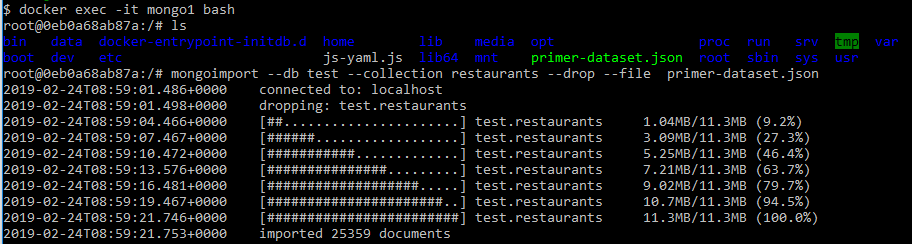
Revisamos que el archivo haya pasado al contenedor mongo1 por medio de la terminal:

**$ docker exec -it mongo1 bash**

**# ls**

Importamos la colección al contenedor mongo1:

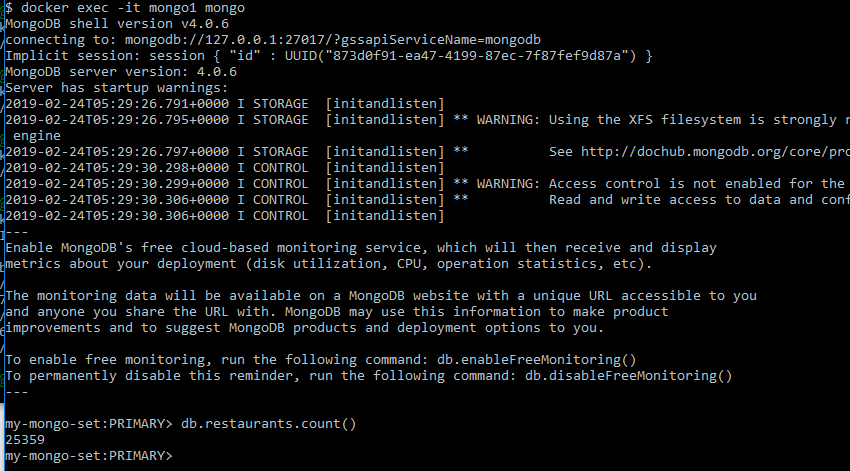
**# mongoimport --db test --collection restaurants --drop --file primer-dataset.json**

****

Posterior al cargue podemos verificar haciendo consultas en los diferentes shell de los contenedores mongo:

**$ docker exec -it mongo1 mongo**

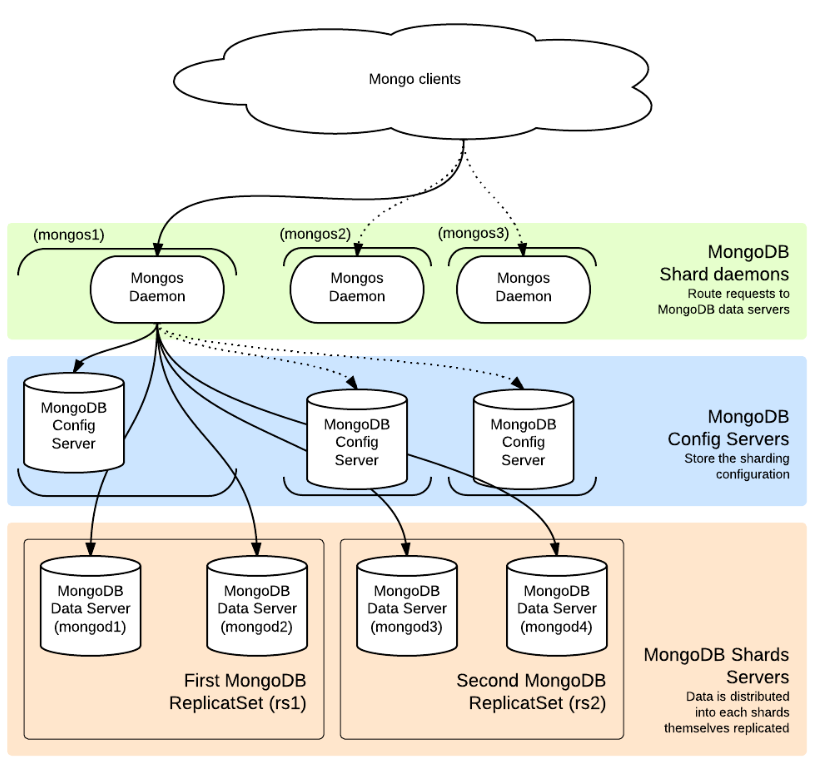
**>db.restaurants.count()**



Verificar también Ingresando a los otros shell:



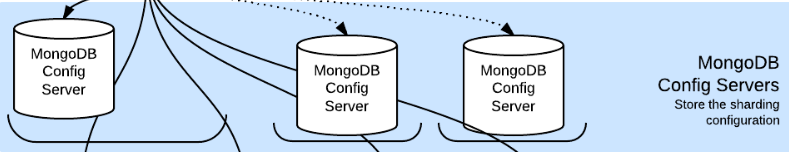
* **Docker - MongoDB: Replica set (Sharding):**

****

1. **Creamos nuestra red:**

**$ docker network create my-mongo-cluster2**

1. **Creamos los servidores de configuración:**

****

**$ docker run -p 26051:26050 --name cfg1 --net my-mongo-cluster mongo mongod --configsvr --replSet configrs --port 26050 --logappend**

**$ docker run -p 26052:26050 --name cfg2 --net my-mongo-cluster2 mongo mongod --configsvr --replSet configrs --port 26050 --logappend**

**$ docker run -p 26053:26050 --name cfg3 --net my-mongo-cluster2 mongo mongod --configsvr --replSet configrs --port 26050 --logappend**

El parámetro “–configsvr” es el que indica que se trata de instancias de configuración. Asimismo el parámetro “–replSet configrs” indica que estas instancias estarán asociadas al conjunto de réplica con nombre “configrs”.

Ingresamos a la consola del primer servidor de configuración:

**$ docker exec -it cfg1 /bin/bash**

Ingresamos al shell del servidor mongo uno de configuración:Activamos la replicación, para ello iniciamos la instancia de mongo en el puerto 26050 (tenemos mismo puerto interno):

**# mongo -port 26050**

Generamos una variable con la configuración de réplica:

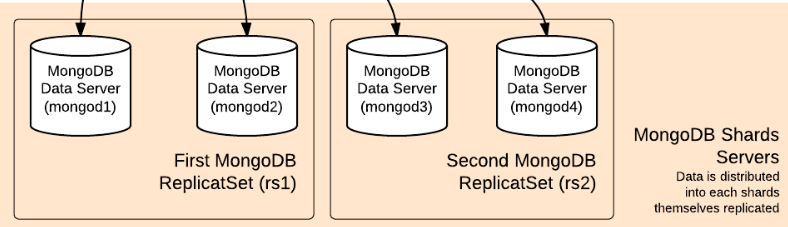
**> conf = {\_id :"configrs","members" : [{\_id : 0, host: "cfg1:26050"}, {\_id: 1,host:"cfg2:26050"}, {\_id:2, host: "cfg3:26050"}]}**

Inicializamos el conjunto de réplica con la configuración:

**> rs.initiate(conf)**

1. **A continuación el turno de los conjuntos réplica:**

Creamos los conjuntos de réplica por cada grupo hay un maestro y un esclavo:



**$ docker run -d --name p1\_1 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p1 --logappend --oplogSize 50**

**$ docker run -d --name p1\_2 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p1 --logappend --oplogSize 50**

**$ docker run -d --name p2\_1 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p2 --logappend --oplogSize 50**

**$ docker run -d --name p2\_2 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p2 --logappend --oplogSize 50**

**$ docker run -d --name p3\_1 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p3 --logappend --oplogSize 50**

**$ docker run -d --name p3\_2 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p3 --logappend --oplogSize 50**

**$ docker run -d --name p4\_1 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p4 --logappend --oplogSize 50**

**$ docker run -d --name p4\_2 --net my-mongo-cluster2 mongo mongod --shardsvr --replSet p4 --logappend --oplogSize 50**

**o**

**$ docker run -d --name p1\_1 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p1**

**$ docker run -d --name p1\_2 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p1**

**$ docker run -d --name p2\_1 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p2**

**$ docker run -d --name p2\_2 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p2**

**$ docker run -d --name p3\_1 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p3**

**$ docker run -d --name p3\_2 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p3**

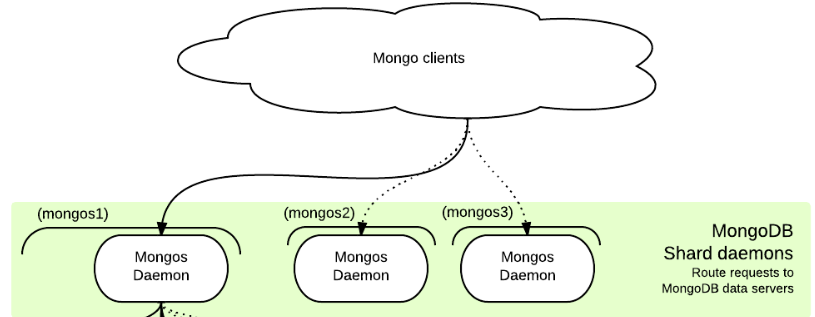
**$ docker run -d --name p4\_1 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p4**

**$ docker run -d --name p4\_2 --net my-mongo-cluster2 --expose=27018 mongo mongod --shardsvr --port 27018 --replSet p4**

La opción **--shardsvr** indica que es un servidor de sharding. Observa que creamos 4 conjuntos réplica con 2 miembros en cada caso.

* **replSet** Indicamos el nombre de la Replica Set a la que va a pertenecer nuestro servicio.
* **dbpath** Será el directorio donde se guardan nuestros datos.
* **logpath <ruta/fichero>** Fichero donde se guardará el log de todo lo que ocurra en el servicio.
* **port** Puerto a utilizar. MongoDB utilizar el puerto 27017 si no especificamos ninguno.
* **fork** Para que el servicio corra en background. Opcional.
* **logappend** Sirve para no vaciar el fichero de log en caso de que ya exista. Opcional.
* **smallfiles** Utilizado en entornos de desarrollo cuando no tenemos necesidad de grandes cantidades de datos.
* **oplogSize <tamaño>** Utilizado en entornos de desarrollo para limitar el tamaño del fichero de operaciones log. Este fichero es usado para poder llevar a cabo la replicación.

1. **Creación de los procesos mongos:**

****

Los procesos mongos son los que harán de balanceadores para la distribución de las queries entre el conjunto que conforma el shard.

**$ docker run -d --name mongos1 -p 26601:27017 --net my-mongo-cluster2 mongo mongos --configdb configrs/cfg1:26050, cfg2:26050, cfg3:26050--logappend**

**$ docker run -d --name mongos2 -p 26602:27017 --net my-mongo-cluster2 mongo mongos --configdb configrs/cfg1:26050, cfg2:26050, cfg3:26050 --logappend**

Le indicamos a mongos los servidores de configuración a través del parámetro “–configdb”; seguido del conjunto de réplica de los servidores de configuración “configrs” y, las direcciones y puertos de los mismos (26050 y 26051) (mongos –configdb configrs/127.0.0.1:26050,127.0.0.1:26051).

El primer mongos no tiene puerto, lo que significa que por defecto conecta con 27017 (puerto tcp por defecto en mongo). Es el puerto por defecto.

Se pueden crear todos los mongos que se quieran; son accesos de los clientes.

1. **Nos conectamos a los nodos y los configuramos:**

**$ docker exec -it p1\_1 /bin/bash**

Iniciamos una instancia de mongo en el puerto **27018**:

**# mongo --port 27018**

Activamos los conjuntos de réplica primarios los cuales se configurarán como primarios y los segundos u otros como secundarios:

**> config = {\_id: "p1",**

**members: [ {\_id:0, host:"p1\_1:27018" },{\_id:1, host:"p1\_2:27018" } ] }**

**rs.initiate(config)**

**> rs.status()**

**$ docker exec -it p2\_1 /bin/bash**

**# mongo --port 27018**

**config = {\_id: "p2",**

**members: [ {\_id:0, host:"p2\_1:27018" },{\_id:1, host:"p2\_2:27018" } ] }**

**rs.initiate(config)**

**> rs.status()**

**$ docker exec -it p3\_1 /bin/bash**

**# mongo --port 27018**

**config = {\_id: "p3",**

**members: [ {\_id:0, host:"p3\_1:27018" },{\_id:1, host:"p3\_2:27018" } ] }**

**rs.initiate(config)**

**> rs.status()**

**$ docker exec -it p4\_1 /bin/bash**

**# mongo --port 27018**

**config = {\_id: "p4",**

**members: [ {\_id:0, host:"p4\_1:27018" },{\_id:1, host:"p4\_2:27018" } ] }**

**rs.initiate(config)**

**> rs.status()**

**Nos conectamos a un cliente:**

Ponemos en marcha las particiones: Para ello conectamos en el cliente sin puerto (mongo):

**$ docker exec -it mongos1 mongo**

Luego indicamos los maestros de cada conjunto de réplica de las instancias de datos:

**> sh.addShard("p1/p1\_1:27018")**

**> sh.addShard("p2/p2\_1:27018")**

**> sh.addShard("p3/p3\_1:27018")**

**> sh.addShard("p4/p4\_1:27018")**

Verificamos el estado:

**> sh.status();**