1. EXPLORANDO KIAR KV

1.1. Instalación en Ubuntu/Mint

Versión y Sitio web: https://www.tiot.jp/riak-docs/riak/kv/3.0.11

1. Instalar dependencias

```
apt-get install libc6 libc6-dev libc6-dbg
```

2. Obtener paquete

```
cd ~/Desktop
mkdir riak
cd riak
wget https://files-source.tiot.jp/riak/kv/3.0/3.0.11/ubuntu/jammy64/riak_3.0.11-OTP22.3_amd64.deb
```

3. Instalar

```
sudo dpkg -i riak_3.0.11-OTP22.3_amd64.deb
```

4. Configurar parámetro del Kernel para aumentar el número de archivos abiertos que pueden existir en el sistema

- Agregar las siguientes líneas en `/etc/security/limits.conf`

```
riak soft nofile 65536
riak hard nofile 200000
```

5. Iniciar el servidor

```
sudo riak start
```

6. Validar la instalación, verificar que se obtenga la siguiente salida: `pong`

```
riak ping
```

7. Obtener los properties de la BD empleando el comando `curl`

```
curl -v http://127.0.0.1:8098/types/default/props
```

1.2. Conceptos básicos

1.2.1. Buckets

- Buckets son empleados para definir un *espacio virtual de almacenamiento* de objetos Riak.
- Un bucket puede considerarse como el equivalente a una tabla en el modelo relacional

1.2.2. Bucket types

- Un bucket puede ser configurado empleando ***bucket types***.
- Un bucket Type permite crear y modificar un conjunto de configuraciones las cuales pueden ser aplicadas a un conjunto de buckets.

Ejemplos de parámetros

Parámetro	Descripción
`n_val`	Número de copias de cada objeto que serán almacenadas en el cluster. Por default es 3.
`last_write_wins`	Indica si el timestamp de un objeto será utilizado para decidir qué cambio es el que se persiste bajo un ambiente de concurrencia (múltiples y simultáneas escrituras).
`precommit, postcommit`	Funciones escritas en Erlang que pueden ser invocadas antes y después de realizar una escritura de un objeto.
`datatype`	En caso de emplear Riak data types, indica el tipo de dato que será empleado en el bucket. Posibles valores: counter, set, map

De forma adicional, es posible crear nuevos bucket types

```
riak admin bucket-type create robert
riak admin bucket-type create earth '{"props":{"n_val":2}}'
riak admin bucket-type create fire '{"props":{"n_val":2}}'
riak admin bucket-type create wind '{"props":{"n_val":2}}'
```

Para que el datatype pueda emplearse posterior a su creación, este debe ser activado.

```
riak admin bucket-type activate <bucket_type>
```

Mostrar el status del bucket type

```
riak admin bucket-type status <bucket_type>
```

Otro uso de un bucket type es agregar un tercer nivel de Jerarquía para organizar e identificar objetos.

Cada objeto puede ser identificado a 3 niveles: type -> bucket -> key. A este tercer nivel que agrega el bucket type se le conoce como ***Additional Namespace.***

1.2.3. Keys

Cadenas empleadas para identificar objetos. Cada bucket representa a un ***keyspace*** diferente. La pareja Key- Objeto se considera como una entidad simple.

1.2.4. Objects

Representan la única unidad de almacenamiento. Un objeto representa a una estructura identificada por un Key. Cada objeto puede estar formado por los siguientes elementos:

- Bucket al que pertenece
- key
- Vector clock
- Metadatos (parejas key - value)

1.3. Modelado de objetos en Riak

- El mejor desempeño se obtiene cuando una aplicación es construida en términos de operaciones CRUD (create, read, update, delete).
- Para localizar a un objeto se consideran 3 niveles llamados locators:
  - La Key del objeto
  - El bucket al que pertenece
  - El tipo de dato del bucket en cual determina la configuración de replicación y otras propiedades.
- Para navegar o explorar el contenido de un bucker, se puede aplicar el concepto de ***nested key/value hash***

Ejemplo:

```
simpsons = {
  'season 1': {
    { 'episode 1': 'Simpsons Roasting on an Open Fire' },
    { 'episode 2': 'Bart the Genius' },
    # ...
  },
  'season 2': {
    { 'episode 1': 'Bart Gets an "F"' },
    # ...
  },
  # ...
}
```

Cada nivel de anidamiento puede definirse por una key. En este ejemplo `Season ->episode`.

- SI se desea obtener los datos de la temporada 2 se escribe lo siguiente:

```
simpsons['season 4']['episode 12']
```

Ambos valores son tratados como keys. A esta técnica se le llama ***lookup operations***. Esta característica es muy rápida ya que no se tiene que hacer búsquedas entre renglones y columnas para encontrar al objeto: `bucket/key address`.

- Estructura de la URL:

```
GET/PUT/DELETE /bucket/<season>/keys/<episode number>
```

El ejemplo anterior puede extenderse con un bucket type:

```
simpsons = {
  'good': {
    'season X': {
      { 'episode 1': '<title>' },
      # ...
    }
  },
  'bad': {
    'season Y': {
      { 'episode 1': '<title>' },
      # ...
    }
  }
}
```

- URL:

```
GET/PUT/DELETE /types/<good or bad>/buckets/<season>/keys/<episode number>
```

- Busqueda

```
simpsons['good']['season 8']['episode 6']
```

1.4. Creando objetos

- Forma básica para escrituras

```
PUT /types/<type>/buckets/<bucket>/keys/<key>
```

Ejemplo

Almacenar un objeto que contiene información de un Perro llamado Rufus con key `rufus`, en el bucket `dogs` con el tipo de bucker `animals`.

- Ejemplo en Java

```
String quote = "WOOF!";
Namespace bucket = new Namespace("animals", "dogs");
Location rufusLocation = new Location(bucket, "rufus");
RiakObject rufusObject = new RiakObject()
  .setContentType("text/plain")
  .setValue(BinaryValue.create(quote));
StoreValue storeOp = new StoreValue.Builder(rufusObject)
  .withLocation(rufusLocation)
   .build();
client.execute(storeOp);
```

- Ejemplo en JavaScript

```
var riakObj = new Riak.Commands.KV.RiakObject();
riakObj.setContentType('text/plain');
riakObj.setValue('WOOF!');
client.storeValue({
  bucketType: 'animals', bucket: 'dogs', key: 'rufus',
  value: riakObj
}, function (err, rslt) {
    if (err) {
        throw new Error(err);
    }
});
```

Ejemplo

Almacenar un objeto JSON

```
curl -v -X POST -d '
  { "clienteId" : 210839028,
    "nombre":  "Diego",
    "codigoPais": "mx"
  }
'
-H "Content-type: application-JSON"
http://localhost:8098/buckets/session/keys/21983we29e
```

1.5. HTTP API

Riak define un API HTTP que permite realizar una gran variedad de operaciones. La lista de URLs puede consultarse en https://www.tiot.jp/riak-docs/riak/kv/3.0.11/developing/api/http

El API puede emplearse haciendo uso del comando `curl`.

El comando `curl` se emplea para transferir datos de o hacia un servidor empleando alguno de los siguientes protocolos: HTTP, FTP, IMAP, POP3, SCP, SFTP, SMTP, TFTP, TELNET, LDAP, o FILE.

1.5.1. Almacenando objetos con HTTP

Estructura del URL

```
POST /types/type/buckets/bucket/keys        # Riak-defined key
PUT /types/type/buckets/bucket/keys/key     # User-defined key
POST /buckets/bucket/keys                   # Riak-defined key
PUT /buckets/bucket/keys/key                # User-defined key
```

Headers obligatorios

- Content-type
- X-Riak-Vclock
- X-Riak-Meta-*
- X-Riak-Index-*

Headers opcionales, sólo para operaciones PUT

- If-None-Match
- If-Match, I
- f-Modified-Since
- If-Unmodified-Since

Query params opcionales

- w
- dw
- pw
- returnbody

Ejemplo sin Key

```
curl -v http://127.0.0.1:8098/buckets/test/keys \
       -H "Content-Type: text/plain" -d 'this is a test'
```

Salida:

```
*   Trying 127.0.0.1:8098...
* Connected to 127.0.0.1 (127.0.0.1) port 8098 (#0)
> POST /buckets/test/keys HTTP/1.1
> Host: 127.0.0.1:8098
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Type: text/plain
> Content-Length: 14
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 201 Created
< Vary: Accept-Encoding
< Server: MochiWeb/2.20.0 WebMachine/1.11.1 (greased slide to failure)
< Location: /buckets/test/keys/7x9bX2FnXkbp5v2x5Qi7DK3jeKz
< Date: Wed, 22 Feb 2023 06:57:07 GMT
< Content-Type: text/plain
< Content-Length: 0
<
* Connection #0 to host 127.0.0.1 left intact
```

Ejemplo con key

```
 curl -v -XPUT -d '{"bar":"baz"}' -H "Content-Type: application/json" -H
"X-Riak-Vclock: a85hYGBgzGDKBVIszMk55zKYEhnzWBlKIniO8mUBAA=="
http://127.0.0.1:8098/buckets/test/keys/doc?returnbody=true
```

1.5.2. Consultar objetos con HTTP

Estructura del URL

```
GET /types/type/buckets/bucket/keys/key
GET /buckets/bucket/keys/key
```

Ejemplo

```
curl -v http://127.0.0.1:8098/buckets/test/keys/doc2
```

1.6. Riak Search

Permite realizar búsquedas empleando el contenido del ***value*** empleando *Lucene Indexes*.
Riak Search representa una integración entre los siguientes 2 elementos:

- Solr: Para realizar indexado y búsquedas haciendo uso del value
- Riak: Empleada para realizar el almacenamiento y distribución de los datos

Existen 3 conceptos importantes para almacenar datos y realizar búsquedas de forma adecuada:

- ***Schemas***: Le permiten a Solr como realizar el indexado de atributos
- ***Índices***: Empleados para realizar búsquedas
- ***Asociación bucket-índice***: Permite notificar a Riak cuando deben aplicarse índices al *value***.**

Riak Search debe ser configurado con un *esquema* de Solr que le permita conocer cómo indexar atributos del *value*. De no existir se emplea un esquema llamado `_yz_default`.

El siguiente paso es crear un índice, el cual representa una colección de datos similares empleados para realizar consultas. Al crear un índice se puede especificar un *esquema*. De no especificarlo, se emplea el esquema por default.

Ejemplo
Creando un índice con el esquema por default.

```
export RIAK_HOST="http://localhost:8098"
curl -XPUT $RIAK_HOST/search/index/famous
```

Para especificar el esquema se emplea la siguiente sintaxis:

```
curl -XPUT $RIAK_HOST/search/index/famous \
     -H 'Content-Type: application/json' \
     -d '{"schema":"_yz_default"}'
```

El último paso para configurar las búsquedas es la asociación del índice ya sea con un *bucket type* (todos los buckets al que se le apliquen este tipo tendrán asociado al índice) o con un bucket específico. Para ambos casos el índice se asocia empleando la propiedad llamada `search_index`.

Ejemplo

- Asociación con un bucket

```
curl -XPUT $RIAK_HOST/types/animals/buckets/cats/props \
     -H 'Content-Type: application/json' \
     -d '{"props":{"search_index":"famous"}}'
```

- Asociación con un bucket type

```
riak admin bucket-type create animals '{"props":{"search_index":"famous"}}'
riak admin bucket-type activate animals
```

1.6.1. Realizar búsquedas

Antes de realizar búsquedas se debe realizar una configuración de seguridad que limita el uso de esquemas e índices.

```
riak admin security grant search.admin on schema to username
riak admin security grant search.admin on index to username
riak admin security grant search.query on index to username
riak admin security grant search.query on index famous to username
```

Ejemplo

- Insertar datos

```
curl -XPUT $RIAK_HOST/types/animals/buckets/cats/keys/liono \
     -H 'Content-Type: application/json' \
     -d '{"name_s":"Lion-o", "age_i":30, "leader_b":true}'

curl -XPUT $RIAK_HOST/types/animals/buckets/cats/keys/cheetara \
     -H 'Content-Type: application/json' \
     -d '{"name_s":"Cheetara", "age_i":28, "leader_b":false}'

curl -XPUT $RIAK_HOST/types/animals/buckets/cats/keys/snarf \
     -H 'Content-Type: application/json' \
     -d '{"name_s":"Snarf", "age_i":43}'

curl -XPUT $RIAK_HOST/types/animals/buckets/cats/keys/panthro \
     -H 'Content-Type: application/json' \
     -d '{"name_s":"Panthro", "age_i":36}'
```

Para realizar el indexado de datos se emplean ***extractors***.

Un extractor acepta un Riak Value de un determinado Content Type y realiza una conversión de una lista de atributos que pueden ser indexados. Existen extractors específicos a cada Content Type.

Para determinar las columnas a indexar se revisan los nombres de los atributos en el documento JSON, XML, etc. Se emplean los siguientes sufijos para identificar a la lista de atributos:

- `_s` representa a un string
- `_i` representa a un número entero
- `_b` representa un dato binario, etc.

Ejemplo de una búsqueda simple

```
curl "$RIAK_HOST/search/query/famous?wt=json&q=name_s:Lion*" | json_pp
```

La respuesta de la consulta anterior no es un Riak Value. En su lugar se obtiene un documento JSON

```
{
  "numFound": 1,
  "start": 0,
  "maxScore": 1.0,
  "docs": [
    {
      "leader_b": true,
      "age_i": 30,
      "name_s": "Lion-o",
      "_yz_id": "default_cats_liono_37",
      "_yz_rk": "liono",
      "_yz_rt": "default",
      "_yz_rb": "cats"
    }
  ]
}
```