FaunaDB una base de datos para todos

Darel Martínez Caballero

Nuestra primera base de datos

Accedemos a la dirección → https://fauna.com/ y nos creamos nuestra cuenta. Podemos utilizar los diferentes métodos de acceso (Google, Netlify, github, etc.)

Una vez tenemos la cuenta creada desde la página → https://dashboard.fauna.com/

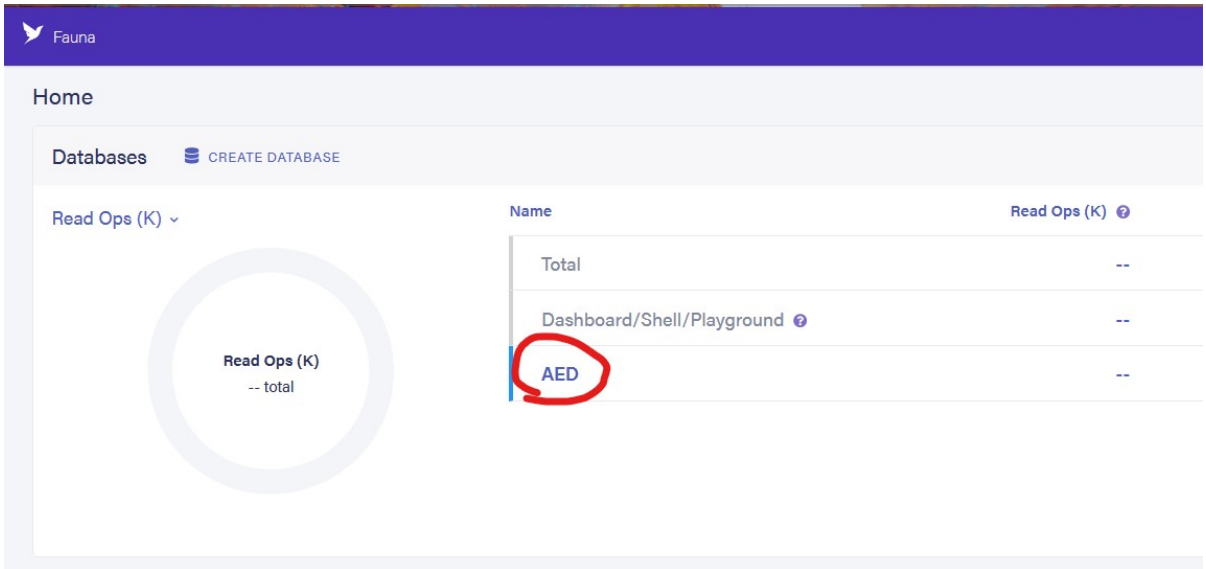
Procedemos a crear nuestra primera base de datos dando click en “Create Database”.

Le ponemos un nombre elegimos la ubicación del servidor.

Nuestra key como rol admin

Una vez hemos creado nuestra primera base de datos podemos crear nuestra key como admin.

Para ello desde https://dashboard.fauna.com/ hacemos click a nuestra base de datos creada.



Ahora que estamos dentro de nuestra base de datos desde el panel lateral podremos dar click en “Security” → new key → elegimos el rol admin y opcionalmente le ponemos un nombre.
Una vez creada aparecerá un mensaje como este:

> ⚠ YOUR KEY'S SECRET IS:
>
> **fnAEdGXzZBAAwFH-jrF4KmxlKAkVkwJYJeTMcN6u**
>
> This secret won't be displayed again, so please copy and save it somewhere safe.

Repositorios maven
Antes de comenzar a trabajar en java debemos agregar al pom.xml de maven los diferentes repositorios necesarios para trabajar con faunadb.

```
<dependencies>
   <dependency>
      <groupId>com.faunadb</groupId>
      <artifactId>faunadb-java</artifactId>
      <version>5.0.0-beta</version>
      <scope>compile</scope>
   </dependency>
   <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>2.0.0-alpha5</version>
   </dependency>
   <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>2.0.0-alpha5</version>
      <scope>test</scope>
   </dependency>
   <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-jdk14 -->
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-jdk14</artifactId>
      <version>2.0.0-alpha5</version>
      <scope>runtime</scope>
   </dependency>
</dependencies>
```

Accediendo como admin y creando una base de datos anidada desde java

Una vez tenemos nuestra key como rol admin y las dependencias cargadas podremos conectarnos desde java.

La forma de realizarlo sería la siguiente:

```
FaunaClient adminClient = FaunaClient.builder()
      .withSecret("fnAEcDG18HAAx2ZqrcLFwVOyjIVVirzokDAngBc3")
      .withEndpoint("https://db.eu.fauna.com:443")
      .build();
```

Para .withEndpoint() dependerá del servidor elegido. En este caso https://db.eu.fauna.com:443

Debemos importar las librerías:

```
import com.faunadb.client.FaunaClient;
```

Ahora que tenemos un cliente con la key de la base de datos especificada podemos crear una nueva base de datos de la siguiente manera

```
Value dbResults = adminClient.query(
        Do(
                If(
                        Exists(Database(DB_NAME)),
                        Delete(Database(DB_NAME)),
                        Value(true)
                ),
                CreateDatabase(Obj("name", Value(DB_NAME)))
        )
).get();

System.out.println("Base de datos creada satisfactoriamente: " + dbResults.at("name").to(String.class).get() + "\n" +
dbResults + "\n");
```

Sin olvidarnos de la key para poder trabajar dentro de esta nueva base de datos.

```
//Generamos la key
Value keyResults = adminClient.query(
        CreateKey(Obj("database", Database(Value(DB_NAME)), "role", Value("server")))
).get();

//Obtenemos la key de la nueva base de datos para acceder y trabajar sobre ella.
String key = keyResults.at("secret").to(String.class).get();
```

Por último realizamos la conexión:

```
//Realizamos la conexión
FaunaClient client = adminClient.newSessionClient(key);
```

Crear una colección

```
//Creamos la colección (Tabla)
Value classResults = client.query(
        CreateCollection(
                Obj("name", Value(CLIENTE_CLASS))
        )
).get();

System.out.println("Crear clase como " + DB_NAME + ":\n " + classResults + "\n");
```

Crear un índice

```
//Creamos un índice para poder recuperar objetos por sus propiedades y no solo por su referencia
Value indexResults = client.query(
      CreateIndex(
            Obj("name", Value(INDEX_NAME) , "source", Collection(Value(CLIENTE_CLASS)))
      )
).get();

System.out.println("Creación de índice para " + DB_NAME + ":\n " + indexResults + "\n");
```

Insertar datos

Si quisiéramos insertar varios datos juntos podríamos optar por:

```
Clientes newCliente6 = new Clientes("555555J", "Hoffman", "Venezolana");
Clientes newCliente7 = new Clientes("555555J", "Hoffman", "Venezolana");

List<Clientes> clientesList = Arrays.asList(newCliente6, newCliente7);

Value clientesListSave = client.query(
      Foreach(
            Value(clientesList),
            Lambda(Value("NXT_CLIENTE"),
                  Create(
                        Collection(Value(CLIENTE_CLASS)),
                        Obj("data", Var("NXT_CLIENTE"))
                  )
            )
      )
).get();
```

Si lo que queremos es insertar un solo dato quizás nos interese mas:

```
Clientes newCliente = new Clientes(codDNIoNIE, nombre, nacionalidad);
Value storeClientResult = client.query(
      Create(
            Collection(Value(CLIENTE_CLASS)),
            Obj("data", Value(newCliente))
      )
).get();
System.out.println("Guardado Cliente:\n " + storeClientResult + "\n");
```

Visualizar datos

Si quisiéramos visualizar los datos de las colecciones

```
Value findAllClientes = client.query(
      SelectAll(Path("data", "data"),
            Map(
                  Paginate(
                        Match(Index(Value(INDEX_NAME)))
                  ),
                  Lambda(Value("NXT_CLIENTE"), Get(Var("NXT_CLIENTE"))))
      )
).get();

Collection<Clientes> allClientesCollections = findAllClientes.asCollectionOf(Clientes.class).get();
allClientesCollections.forEach(nextData -> System.out.println(" " + nextData));
```

Si quisieramos visualizar los ids de referencia

```
Value queryResultIds = client.query(
      SelectAll(Path("data", "id"),
            Paginate(
                  Match(Index(Value(INDEX_NAME)))
            ))
).get();
Collection<String> allClientesIdsCollections = queryResultIds.asCollectionOf(String.class).get();
allClientesCollections.forEach(nextId -> System.out.println(" " + nextId));
```

Borrar datos

Si queremos borrar un dato necesitamos su numero de referencia.

```
System.out.println("Eliminado = " +
      client.query(
            Delete(Ref(Collection("clientes"), Value(ref)))
      ).get());
```

Actualizar datos

Si queremos actualizar algún dato:

```
System.out.println(
      dato + " actualizado = "+
            client.query(
                  Update(
                        Ref(Collection("clientes"), Value(ref)),
                        Obj(
                              "data", Obj(
                                    dato, Value(datoUpdate)
                              )
                        )
                  )
            ).get());
```

Borrar una base de datos

Si queremos borrar una base de datos:

```
Value result = adminClient.query(
        If(
                Exists(Database(dbName)),
                Delete(Database(dbName)),
                Value(true)
        )
).get();

System.out.println(result);
```

Capturar errores
La clase de FaunaCliente nos proporciona información sobre los diferentes posibles errores que se puedan producir durante la ejecución del programa. Aquí están descrito todos los posibles errores de FaunaClient.

```
try {

} catch (BadRequestException bre) {
    System.out.println("Esta consulta no puede ser evaluada");
} catch (InternalException ie) {
    System.out.println("HTTP 500 (Error interno del servidor) al realizar una solicitud a FaunaDB.");
} catch (NotFoundException nfe) {
    System.out.println("error HTTP 404 (No encontrado).");
} catch (PermissionDeniedException pde) {
    System.out.println("error HTTP 403 (Prohibido) desde FaunaDB.");
} catch (StreamingException se) {
    System.out.println("Evento de error en la transmisión.");
} catch (TransactionContentionException tce) {
    System.out.println("Error HTTP 409 desde FaunaDB.");
} catch (UnauthorizedException ue) {
    System.out.println("Error HTTP 401 (no autorizado).");
} catch (UnavailableException ue) {
    System.out.println("El host de FaunaDB no está disponible por algún motivo.");
} catch (UnknownException ue) {
    System.out.println("El cliente desconoce o no puede analizar una respuesta de FaunaDB.");
} catch (ExecutionException ee) {
    System.out.println("Error en tiempo de ejecución");
}
```