

# **Taller NoSQL**

## **Laboratorio 2**

### **Tecnólogo en Informática**

#### **Sede Buceo**

### **Turno Nocturno 2021**

Integrantes:

- Edwin Pistón
- Julio Espinosa
- Maximiliano Nicoletta

Docente: Andrés Pastorini

## Índice

Base de datos seleccionada y fundamento	3
Testeo de endpoints	3
Construcción de la solución	4
Pruebas de Carga y Comportamiento	6
Casos de prueba	8
Observaciones finales sobre las pruebas	9
Pruebas de Automatización con Jenkins	9
Instalación de Jenkins	9
Configuración de Jenkins	10
Instalacion y configuracion del Plugins de NodeJs	11
Creación Tarea Jenkins	12
Salida de la tarea Programada	14
Resultado de las pruebas independientes	15

## Base de datos seleccionada y fundamento

Decidimos utilizar Couchbase como base NoSQL para el Laboratorio 2 dado que fue la herramienta que presentamos en el primer laboratorio, y hallamos en ella facilidad de utilización al asemejarse en su lenguaje (N1QL) al lenguaje utilizado en bases de datos relacionales (SQL). A su vez nos pareció interesante evaluar la eficiencia de esta base de datos ante grandes cargas, evaluando su verdadera capacidad y prestaciones frente a competidores como MongoDB, que son más conocidos en el mercado.

El desarrollo del middleware lo presentamos en .NET dado que la experiencia con la que contamos en este lenguaje nos permite realizar un producto estable, y fácilmente testeable mediante el complemento de Swagger. El servicio cuenta con llamadas POST para el ingreso de nuevos datos, PUT y DELETE para modificación de datos existentes, y GET para la obtención de datos del sistema.

Proyecto público en GIT: <https://github.com/EdwinpistonC/nosql-api>

## Testeo de endpoints

Se presenta en Postman (adjunto protocolo de conexiones) y Swagger, accesible a través de <http://localhost:5001/swagger> una vez ejecutado el proyecto.

Se presentan ilustraciones de los productos accesibles.

The screenshot displays the Swagger UI for the CouchbaseWebAPI. The main header shows the Swagger logo and the API title 'CouchbaseWebAPI 1.0 OAS3'. Below the header, the endpoints are organized into two sections: 'Code' and 'User'. The 'Code' section lists three endpoints: GET /Code, POST /Code, and GET /Code/getCodeById/{id}. The 'User' section lists four endpoints: POST /User, POST /User/authUser, PUT /User/addRol, and DELETE /User/removeRols. On the right side, there is a sidebar with a tree view showing the API structure. The root is 'Couchbase', which contains two collections: 'User' and 'Code'. The 'User' collection has four endpoints: POST User, POST authUser, PUT addRol, and DEL removeRol. The 'Code' collection has three endpoints: GET Code, POST Code, and GET getCodeById.

## Construcción de la solución

Será necesario para utilizar este desarrollo instalar el servidor Couchbase de manera local sobre Docker:

1. En la terminal de Windows ejecutamos:

```
docker run -d --name db -p 8091-8096:8091-8096 -p 11210-11211:11210-11211 couchbase
```

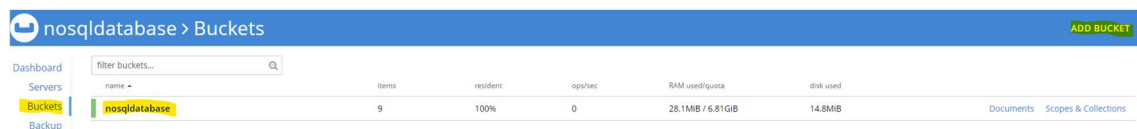
2. Luego se realizará la instalación por defecto accediendo a <http://localhost:8091/> creando el usuario administrador con las credenciales:

Usuario: **adm**

Contraseña: **adm123**

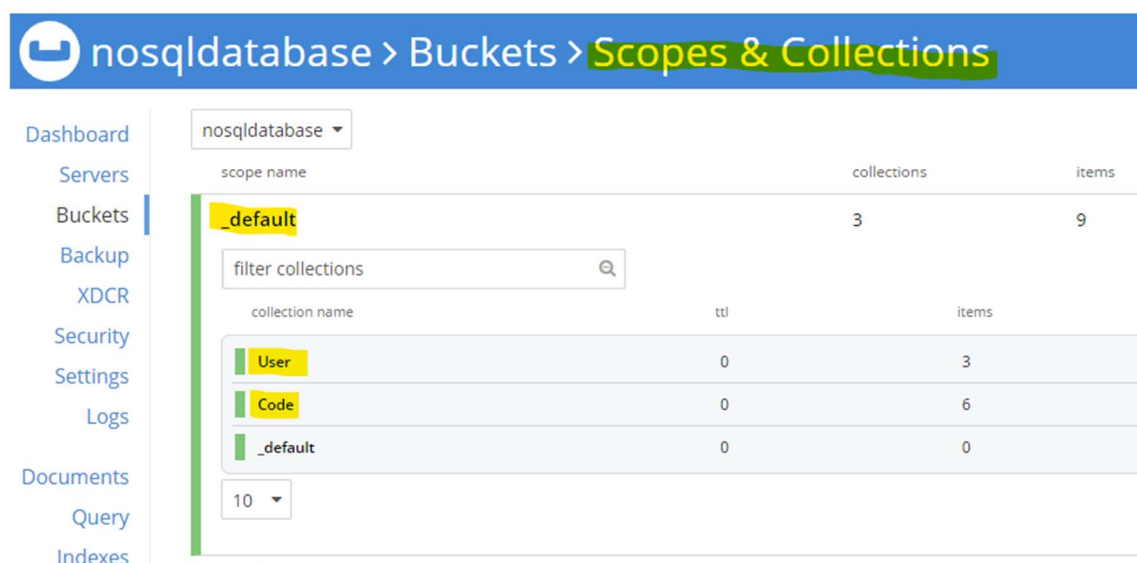
*\*Estos valores son estáticos en el código de manera demostrativa.*

3. Será necesario crear el Bucket con el nombre "**nosqldatabase**" con los valores por defecto



nosqldatabase > Buckets							ADD BUCKET
filter buckets...	name	items	resident	ops/sec	RAM used/quota	disk used	
	nosqldatabase	9	100%	0	28.1MiB / 6.81GiB	14.8MiB	Documents Scopes & Collections

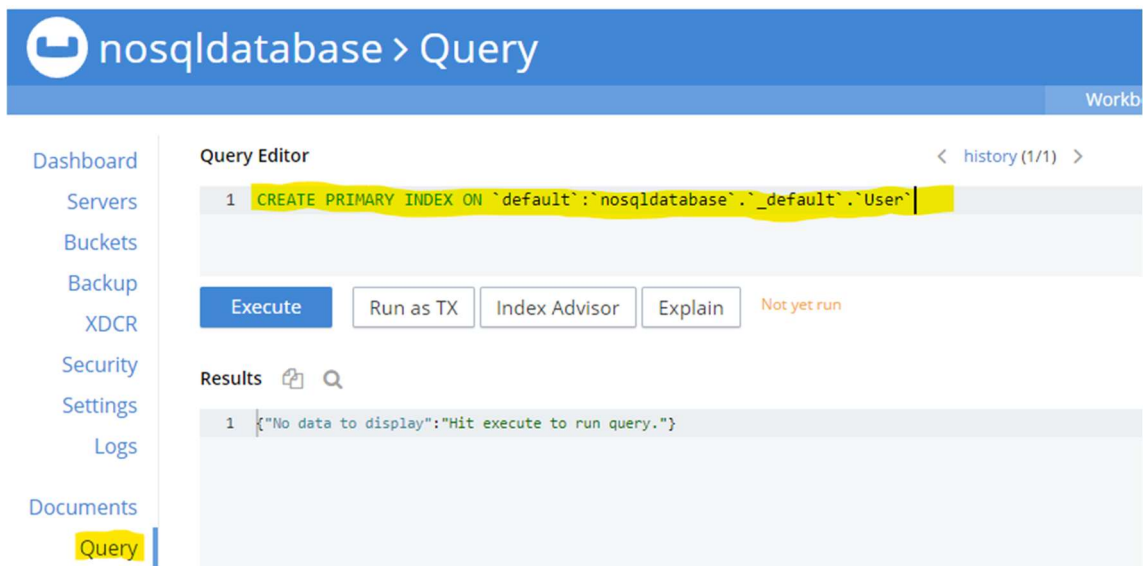
4. Dentro del bucket creado, nos dirigimos a "Scopes & Collections" y dentro del item "\_default" creamos los Scopes:
  - a. User
  - b. Code



nosqldatabase > Buckets > Scopes & Collections			
nosqldatabase			
scope name	collections	items	
_default	3	9	
filter collections			
collection name	ttr	items	
User	0	3	
Code	0	6	
_default	0	0	
10			

5. Desde el Query editor creamos los índices:

- `CREATE PRIMARY INDEX ON `default`:`nosqldatabase`.`_default`.`User``
- `CREATE PRIMARY INDEX ON `default`:`nosqldatabase`.`_default`.`Code``



6. Debemos compilar la solución .NET desde Visual Studio (Código en GIT) o ejecutando el .EXE (Se publica archivo compilado en .ZIP)

#### Dependencias del ejecutable:

- .NET Core 3.1 Runtime (LTS)

#### Dependencias del código:

- Visual Studio 2022
- .NET SDK 5.0

7. Accedemos al sitio de Swagger (<https://localhost:5001/swagger>) o trabajamos desde Postman con los endpoints facilitados.

8. Agregaremos los códigos de error al bucket "Code" para que luego los errores sean reportados cuando gestiono los usuarios. Se podrá realizar tanto desde Postman como desde Swagger

POST /Code

Parameters

No parameters

Request body

application/json

Example Value | Schema

```
{
  "id": 0,
  "description": "string"
}
```

Responses

Code	Description	Links
200	Success	No links

El código que se ingresa será como el que se muestra a continuación para cada uno de los errores requeridos (101, 102, 103, 104)

```
[
  {
    "id": 101,
    "description": "El usuario ya existe"
  }
]
```

## Pruebas de Carga y Comportamiento

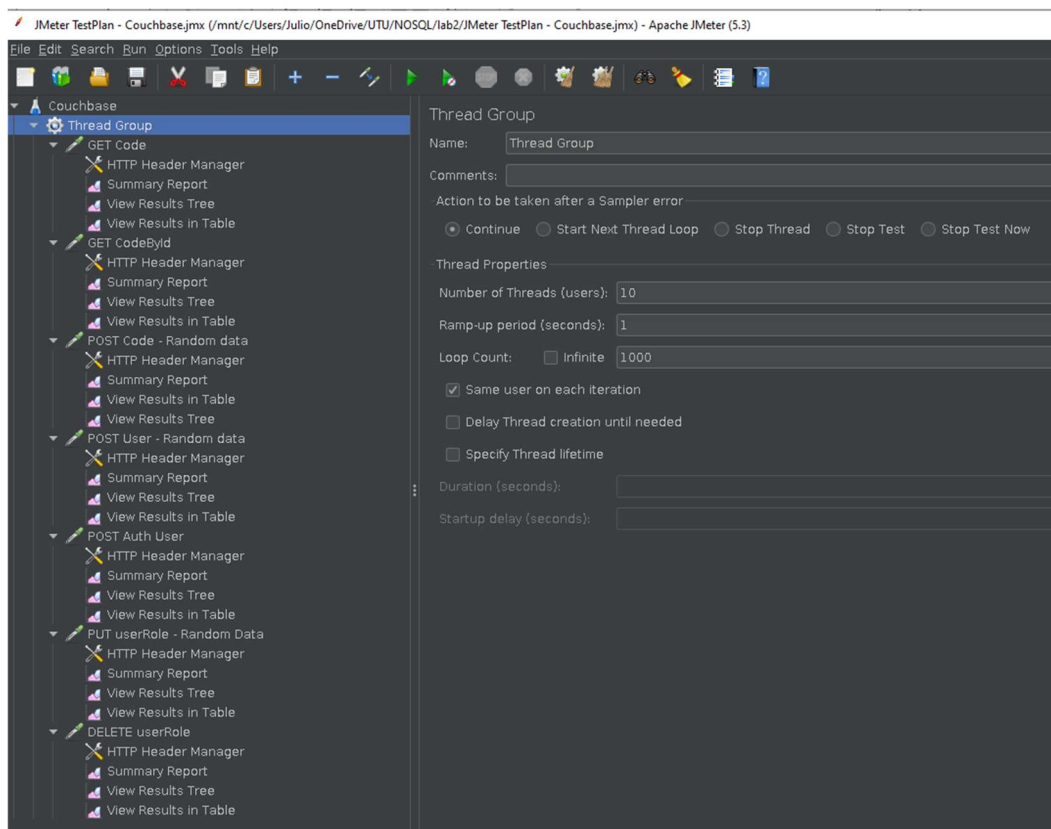
Utilizaremos Apache JMeter 5.3 para realizar la prueba de carga y comportamiento. Dado que el software corre sobre Linux y nuestra instalación se realizó sobre Windows 10 Pro, utilizaremos WSL 2 (<https://docs.microsoft.com/en-us/windows/wsl/install>) con Ubuntu 20.04 LTS para su ejecución. Para la interfaz gráfica de la aplicación utilizamos VcXsrv (<https://ubunlog.com/vcxsrv-nos-permite-usar-apps-de-linux-con-interfaz-de-usuario-en-windows-10/>).

Dentro de JMeter generamos un plan de pruebas de todos los endpoints con carga de información aleatoria para los casos: POST Code (Publicar códigos de error), POST User (Publicar usuarios nuevos) y PUT addRol (agregar roles a un usuario).

El archivo de pruebas se publica junto a este documento.

## Consideraciones de las pruebas

1. Las pruebas se corren localmente por lo que no estamos evaluando carga de la red.
2. Todas las pruebas son realizadas con un solo nodo, lo que no es esperado en un despliegue estándar de Couchbase.
3. La ejecución en paralelo de todas las pruebas podría estar afectando el rendimiento de las pruebas.
4. La interfaz gráfica y recolección de evidencia deteriora el rendimiento de las pruebas, no obstante, a modo ilustrativo y por simplicidad, permaneceremos en esta modalidad.
5. Agregar Códigos de error hace crecer la carga de respuesta al obtener todos los códigos de error, lo que repercute en su rendimiento de forma directa.
6. El *throughput* son las solicitudes por unidad de tiempo
7. Las pruebas son realizadas en un equipo con procesador AMD Ryzen 3 3200G 3600Mhz, 16 GB RAM y disco de estado sólido.



## Casos de prueba

Se realizan pruebas con una muestra de 10000 casos repartidos en 10 hilos de procesamiento. Mayor throughput es mejor.

### Prueba 1

#### HTTP

Request	Path	Throughput
GET	Code	2,8/seg
GET	CodeById	2,8/seg
POST	Code	2,8/seg
POST	User	2,8/seg
POST	authUser	2,8/seg
PUT	userRole	2,8/seg
DELETE	userRole	2,8/seg

Observación: El tiempo de respuesta repercute entre todos los testeos

### Prueba 2

#### HTTP

Request	Path	Throughput	Sample Time (ms)
GET	Code	11,9/seg	~ 810

### Prueba 3

#### HTTP

Request	Path	Throughput	Sample Time (ms)
POST	User	11,6/seg	~ 950

### Prueba 4

#### HTTP

Request	Path	Throughput	Sample Time (ms)
PUT	userRole	4,8/seg	~ 2100

Observación: Se observaron errores del servidor (Codigo de Respuesta 500) debido a la ejecución de 10 hilos en paralelo sobre un mismo usuario. Los errores desaparecen al ejecutar en un único hilo, manteniendo el rendimiento reportado anteriormente

### Prueba 5

#### HTTP

Request	Path	Throughput	Sample Time (ms)
DELETE	userRole	6,4/seg	~ 1500



## Observaciones finales sobre las pruebas

Creamos casos de prueba en los que se generaron nuevas entradas únicas en el servidor y evaluamos la velocidad de respuesta para todos los casos presentados.

Dado que trabajamos en un solo nodo local la velocidad de las pruebas es consistente para todos los métodos, y no se observa el comportamiento expresado en el laboratorio 1, donde las solicitudes PUT/GET tomarán más tiempo cuantos más nodos existan.

Es notoria la mejor velocidad de respuesta en solicitudes GET/POST frente a solicitudes PUT/DELETE.

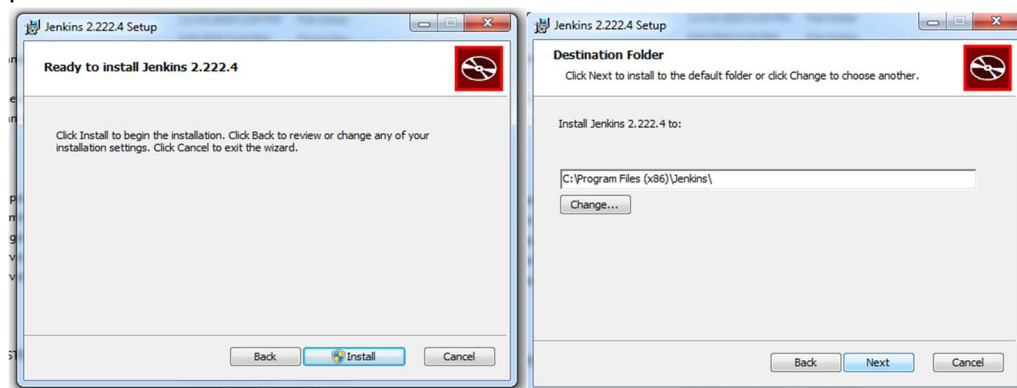
## Pruebas de Automatización con Jenkins

### Requisito previo

- JDK8 +
- Postman 9.0.0 +

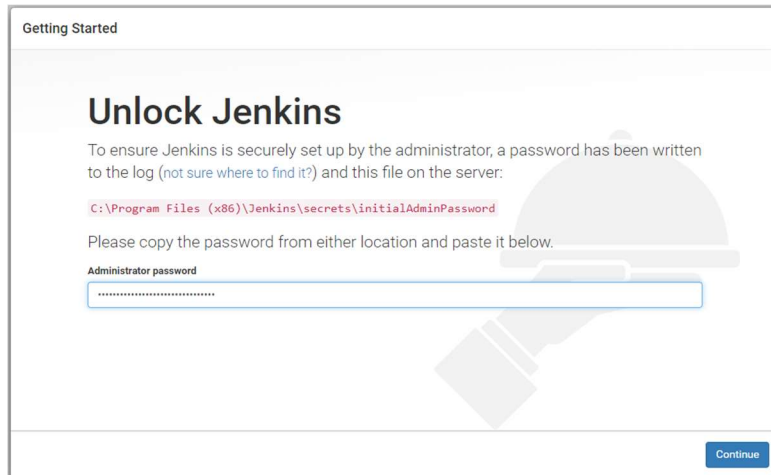
### Instalación de Jenkins

1. Descargar el último paquete de Jenkins MSI para Windows:  
<https://www.jenkins.io/download/>
2. Instalamos el software con Next, Next, Install. Mantenemos la ruta de instalación predeterminada.

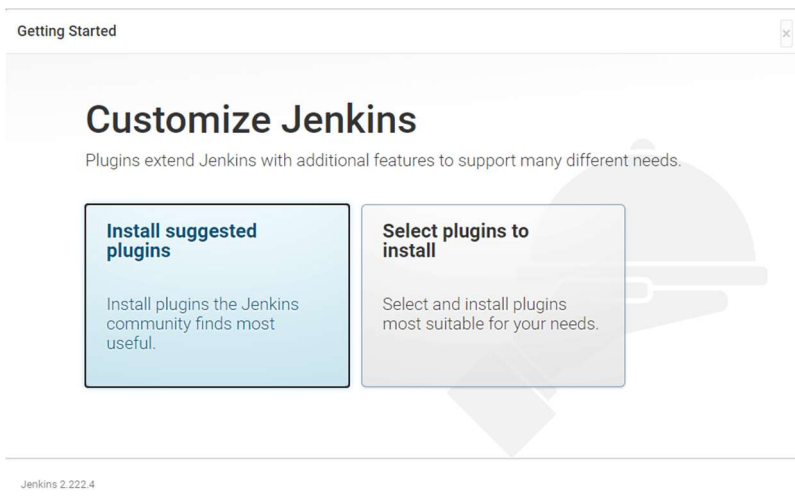


## Configuración de Jenkins

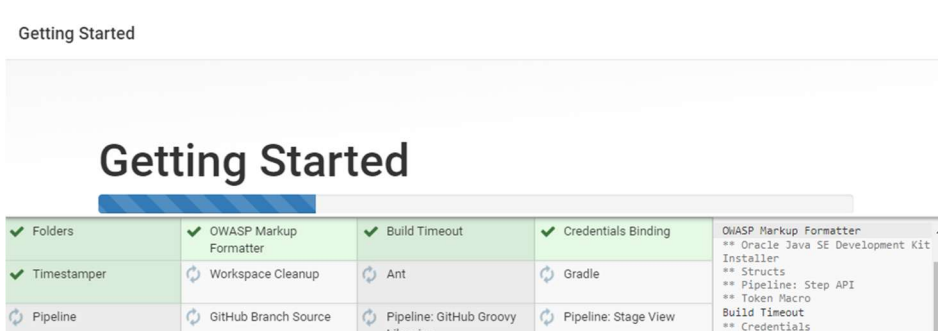
1. Una vez que Jenkins esté instalado, se ejecutará en el puerto 8080, Configuraremos el servicio navegando a <http://localhost:8080> en el navegador.
2. Para desbloquear Jenkins, copie la contraseña de administrador del archivo ubicado en C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword (Considerando que la ruta de instalación fue la predeterminada)



3. Puede instalar los complementos sugeridos o puede seleccionar complementos según su caso de uso. Aquí, instalaremos los complementos sugeridos.

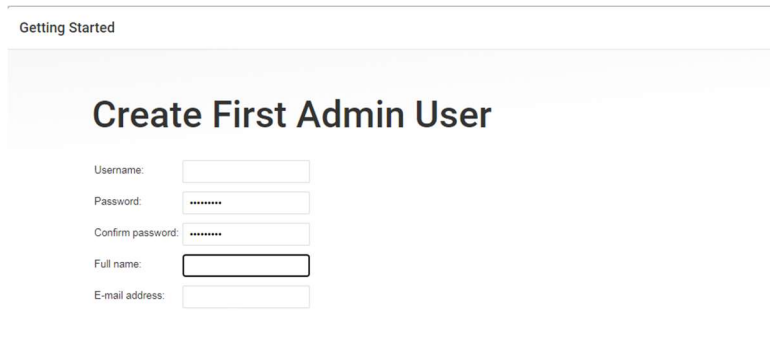


4. Espere hasta que se instalen todos los complementos. Cuando finalice la instalación, haga clic en Continúe.



5. Cree un perfil de administrador para Jenkins. Ingrese los detalles requeridos y haga clic en Guardar y continuar.

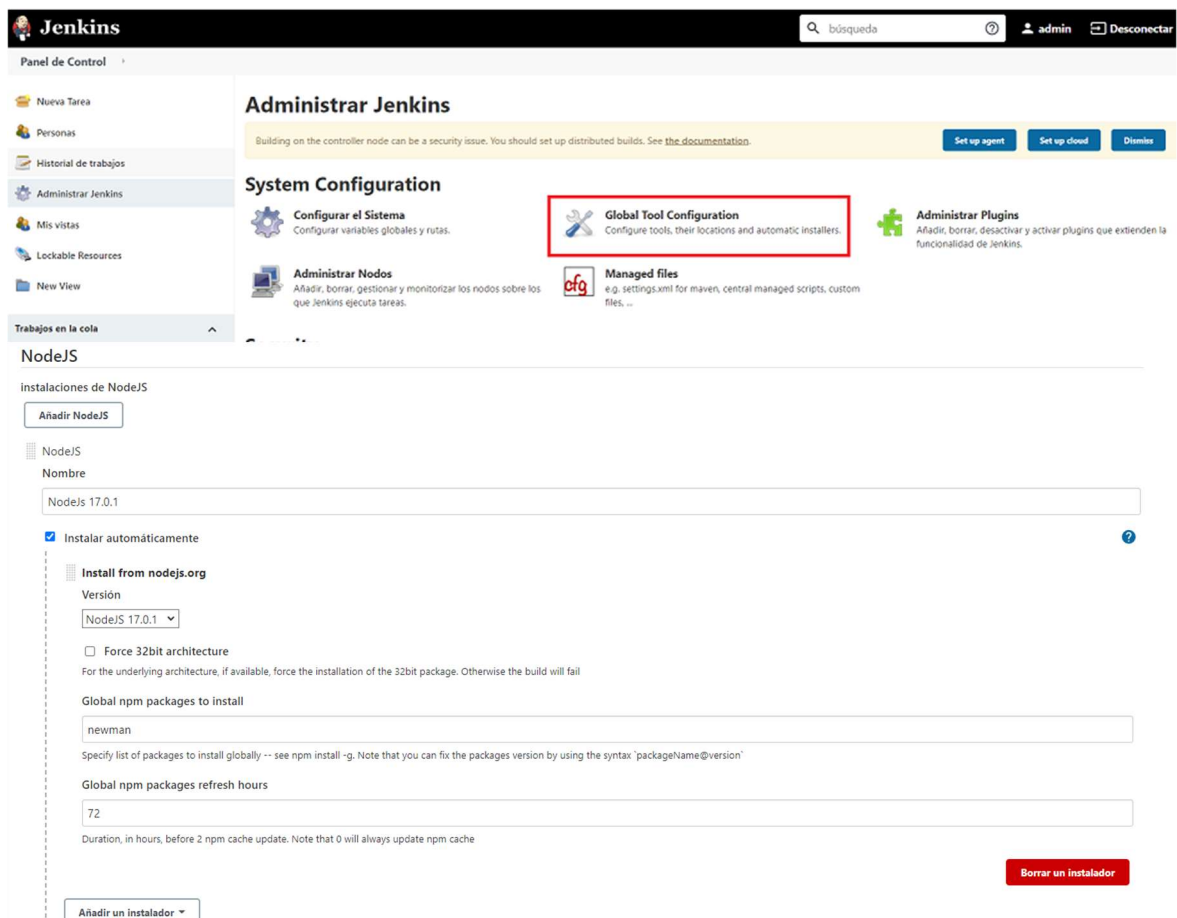
Para nuestro caso  
usuario: **admin**  
contraseña: **admin**.



The screenshot shows the 'Getting Started' page in Jenkins with the 'Create First Admin User' form. The form includes fields for Username, Password, Confirm password, Full name, and E-mail address. The Password and Confirm password fields are masked with dots.

## Instalacion y configuracion del Plugins de NodeJs

1. Instalar paquete descargado de <https://plugins.jenkins.io/nodejs> en Jenkins
2. En "Global tool Configuration" agregamos Nodejs y lo instalamos agregando comando "newman"



The screenshot shows the Jenkins 'Global Tool Configuration' page for NodeJS. The page is titled 'NodeJS' and shows the 'Instalaciones de NodeJS' section. A red box highlights the 'Global Tool Configuration' link in the 'System Configuration' section. The 'NodeJS' section shows the 'Nombre' field set to 'NodeJS 17.0.1'. The 'Instalar automáticamente' checkbox is checked. Under 'Install from nodejs.org', the 'Versión' dropdown is set to 'NodeJS 17.0.1'. The 'Global npm packages to install' field contains 'newman'. The 'Global npm packages refresh hours' field is set to '72'. A red button labeled 'Borrar un instalador' is visible at the bottom right.

## Creación Tarea Jenkins

1. En el panel lateral seleccionamos “Nueva Tarea”



2. Ingresamos el nombre de la tarea, en nuestro caso la llamaremos “NoSql” y seleccionamos la opción Crear un Proyecto de Estilo Libre.

Enter an item name

NoSql

**Crear un proyecto de estilo libre**  
Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Crear un proyecto multi-configuración**  
Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.

**Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

**Multibranch Pipeline**  
Creates a set of Pipeline projects according to detected branches in one SCM repository.

3. Agregamos como origen del código fuente desde el proyecto público en Git

<https://github.com/EdwinpistonC/nosql-api>

Configurar el origen del código fuente

☐ Ninguno

☒ Git

Repositories

Repository URL

https://github.com/EdwinpistonC/nosql-api

Credentials

- none - Add

Avanzado...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

\*/main

Add Branch

4. En la pestaña “Entorno de ejecución” marcamos la opción “Provide Node & npm bin/ folder to PATH”. Luego en la parte de ejecutar seleccionamos la opción “Ejecutar un comando de Windows” y escribimos el comando
- “newman run + {dirección del archivo de pruebas Postman}”**

(el mismo se entrega de manera conjunta a este documento).

The screenshot shows the Newman configuration window with the 'Entorno de ejecución' (Execution Environment) tab selected. The 'Acciones para ejecutar después' (Actions to run after) section has several checkboxes, with 'Provide Node & npm bin/ folder to PATH' checked. Below this, the 'NodeJS Installation' dropdown is set to 'NodeJS 17.0.1'. The 'npmrc file' dropdown is set to '- use system default -'. The 'Cache location' dropdown is set to 'Default (~/.npm or %APP\_DATA%\npm-cache)'. The 'With Ant' checkbox is unchecked. The 'Ejecutar' (Execute) section is active, showing the option 'Ejecutar un comando de Windows' (Execute a Windows command). The command field contains the text: `newman run D:\Estudio\Obligatorio2021.postman_collection.json --insecure`.

## Salida de la tarea Programada

Running as SYSTEM

Ejecutando en el espacio de trabajo

C:\WINDOWS\system32\config\systemprofile\AppData\Local\Jenkins\.jenkins\workspace\NoSql

The recommended git tool is: NONE

No credentials specified

> git.exe rev-parse --resolve-git-dir

C:\WINDOWS\system32\config\systemprofile\AppData\Local\Jenkins\.jenkins\workspace\NoSql\.

Fetching changes from the remote Git repository

> git.exe config remote.origin.url https://github.com/EdwinpistonC/nosql-api # timeout=10

Fetching upstream changes from https://github.com/EdwinpistonC/nosql-api

> git.exe --version # timeout=10

> git --version # 'git version 2.31.1.windows.1'

> git.exe fetch --tags --force --progress -- https://github.com/EdwinpistonC/nosql-api +refs/heads/\*:refs/remotes/origin/\* # timeout=10

> git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10

Checking out Revision eb5b1991982aac5a706b3b3d1050986925ba2dd7 (refs/remotes/origin/main)

> git.exe config core.sparsecheckout # timeout=10

> git.exe checkout -f eb5b1991982aac5a706b3b3d1050986925ba2dd7 # timeout=10

Commit message: "dejando como estaba"

> git.exe rev-list --no-walk eb5b1991982aac5a706b3b3d1050986925ba2dd7 # timeout=10

[NoSql] \$ cmd /c call C:\WINDOWS\TEMP\jenkins1444364734628826335.bat

C:\WINDOWS\system32\config\systemprofile\AppData\Local\Jenkins\.jenkins\workspace\NoSql>newman run D:\Estudio\Obligatorio2021.postman\_collection.json --insecure

newman

## Resultado de las pruebas independientes

### → Creo Usuario Correcto(200)

POST https://localhost:5001/User [200 OK, 159B, 1499ms]

✓ Status Code is 200

### → Creo Usuario Existente (101)

POST https://localhost:5001/User [200 OK, 197B, 46ms]

✓ Status Code is 200

### → Agrego Rol Usuario No Existente (102)

PUT https://localhost:5001/User/addRol [200 OK, 195B, 25ms]

✓ Status Code is 200

### → Agrego Rol Usuario Contraseña invalida (104)

PUT https://localhost:5001/User/addRol [200 OK, 197B, 16ms]

✓ Status Code is 200

### → Agrego Rol Usuario (200)

PUT https://localhost:5001/User/addRol [200 OK, 155B, 33ms]

✓ Status Code is 200

### → Elimino Rol usuario no Existente (102)

DELETE https://localhost:5001/User/removeRols [200 OK, 195B, 20ms]

✓ Status Code is 200

### → Elimino Rol usuario contraseña invalida (104)

DELETE https://localhost:5001/User/removeRols [200 OK, 197B, 15ms]

✓ Status Code is 200

### → Elimino Rol usuario (200)

DELETE https://localhost:5001/User/removeRols [200 OK, 157B, 12ms]

✓ Status Code is 200

### → Elimino Rol no existente usuario (103)

DELETE https://localhost:5001/User/removeRols [200 OK, 219B, 20ms]

✓ Status Code is 200

	executed	failed	
iterations	1	0	
requests	9	0	
test-scripts	9	0	
prerequest-scripts	0	0	
assertions	9	0	
total run duration: 2.4s			
total data received: 357B (approx)			
average response time: 187ms [min: 12ms, max: 1499ms, s.d.: 463ms]			

Finished: **SUCCESS**