



# ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

**INNOVACIÓN PARA LA EXCELENCIA**

# ARQUITECTURA DE CONTENEDORES Y ETL: INTEGRACIÓN EFICIENTE DE BASES DE DATOS Y APIS EN UN ENTORNO DOCKERIZADO

# INTEGRANTES:

- Jacome Ivonne
  - Rodriguez Danny
  - Salazar Daniel



# INTRODUCTION

Tenemos como objetivo describir la arquitectura de un sistema basado en contenedores y su integración con bases de datos, APIs y procesos de extracción, transformación y carga de datos (ETL).

Esta estructura de contenedores y la configuración de la arquitectura permiten asegurar una mayor escalabilidad, modularidad y disponibilidad del sistema, así como una gestión eficiente de los datos.

El uso de contenedores dockerizados facilita la portabilidad y el despliegue de los componentes del sistema, mientras que la replicación de las bases de datos garantiza la redundancia y la tolerancia a fallos.



# Partes que contiene:

## ETL

Consiste en la extracción de datos de diversas fuentes, su transformación para adaptarlos a un formato y estructura común, y finalmente cargarlos en un destino, como una base de datos.  
**Tres etapas principales del proceso ETL:**

- Extracción:** los datos se extraen de múltiples fuentes, como bases de datos, aplicaciones en la nube, entre otros.
- Transformación:** Una vez que los datos se han extraído, se someten a diversas transformaciones y limpiezas para garantizar su calidad y coherencia.
- Carga:** los datos transformados se cargan en un destino adecuado, se definen los esquemas y estructuras de la base de datos de destino y se insertan los datos transformados en ella.

## DOCKERIZADO

Es el proceso de adaptar una aplicación o sistema existente para que se ejecute en contenedores, lo que proporciona portabilidad, escalabilidad y una gestión más eficiente de los recursos.



## SHARDING

Consiste en dividir una base de datos grande en fragmentos más pequeños llamados "shards" o "fragmentos", y distribuir esos fragmentos en varios servidores.

- Facilita la escalabilidad horizontal.**

## REPLICAS

Significa crear y mantener copias exactas de una base de datos original en uno o más servidores adicionales, se mantienen sincronizadas con la base de datos original mediante.

## API

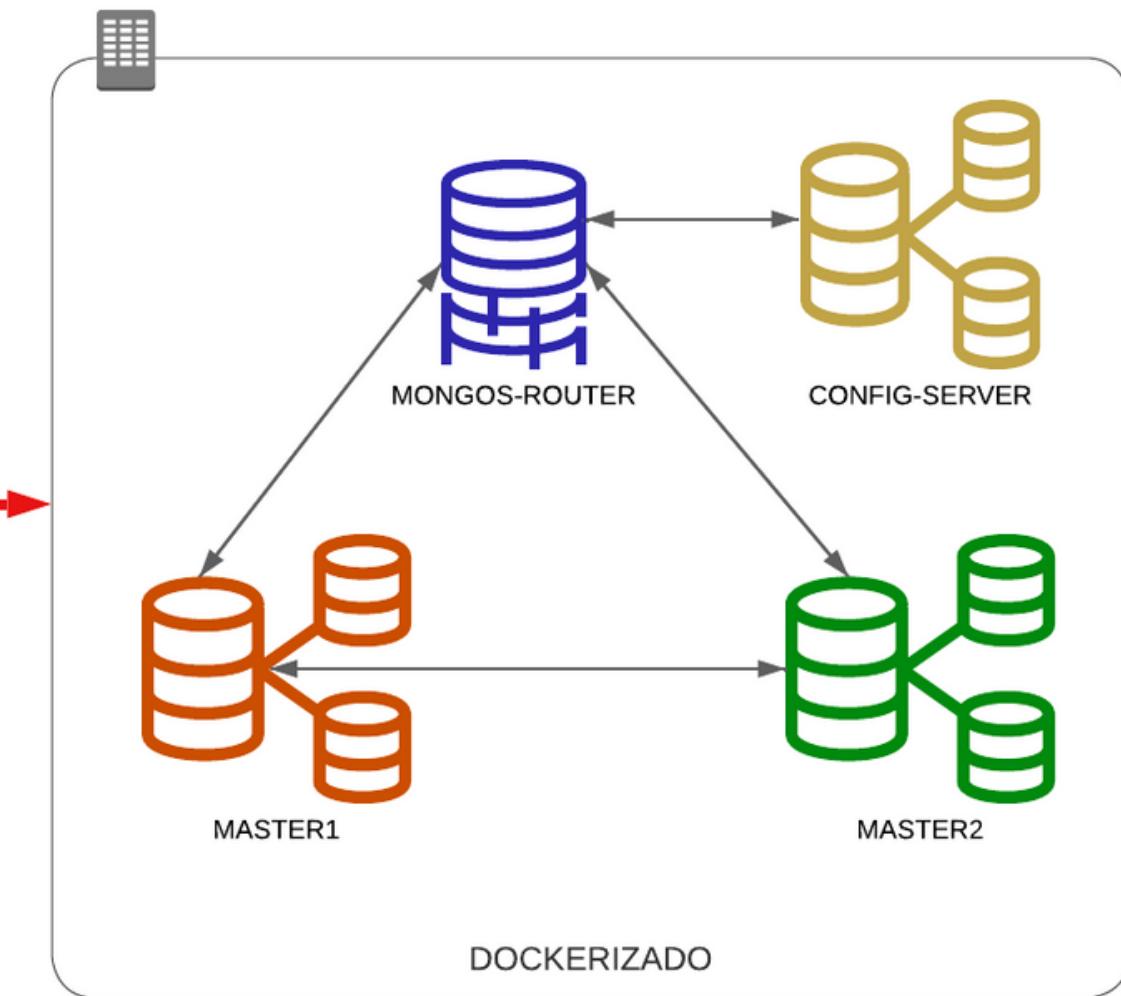
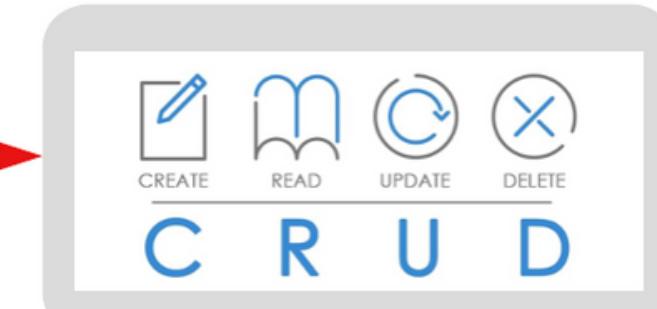
Es una interfaz que define cómo los diferentes elementos de un sistema de software deben interactuar entre sí.

**Node.js:** es un entorno de tiempo de ejecución de JavaScript que permite ejecutar código JavaScript en el lado del servidor.

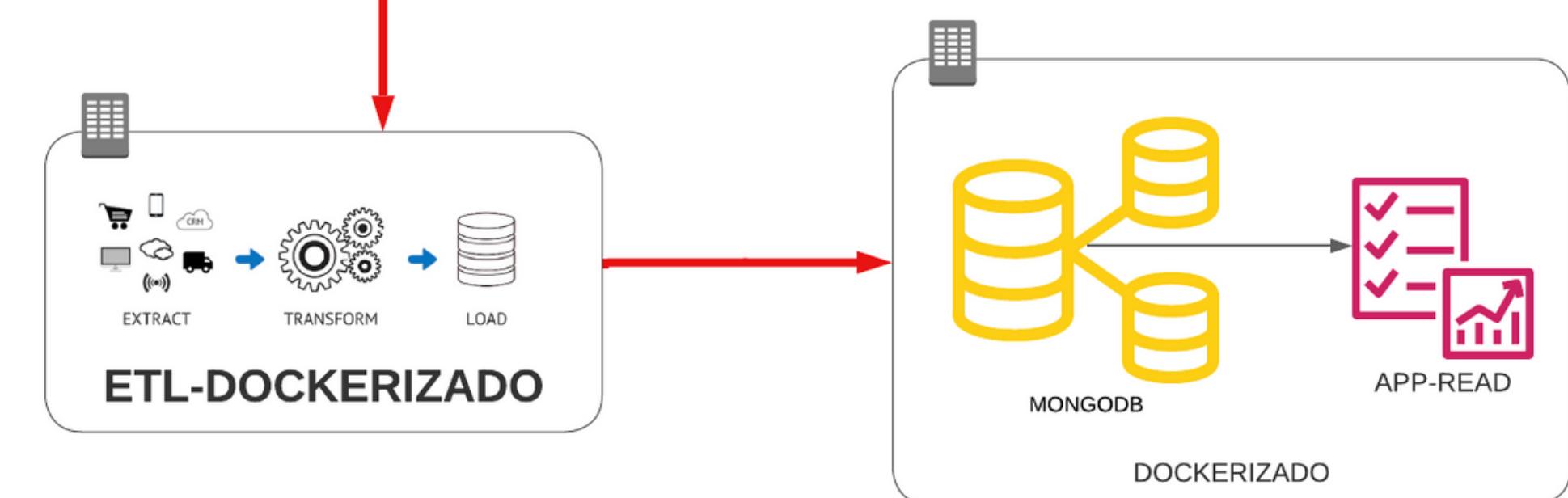
**Express:** Proporciona una capa de abstracción sobre Node.js que simplifica la creación de aplicaciones web y APIs, puedes utilizar Express para crear las rutas y los controladores de la API.

**MongoDB:** es una base de datos NoSQL orientada a documentos.

API



# TOPOLOGIA LOGICA



# CONFIGURACION:

## CONFIG-SERVER/ MASTER1/MASTER2

Con el siguiente comando vamos a inicializar el contenedor donde vamos a tener los tres instancias de MongoDB y a realizar sus replicas.

Entramos al Shell de Mongo del Contenedor:

`mongosh mongodb://192.168.100.85:27019`

Una vez dentro mandamos a inicializar las replicas:

```
CONFIGSVR
rs.initiate(
{
  _id: "configserver",
  configsvr: true,
  members: [
    { _id : 0, host : "192.168.100.85:27019" },
    { _id : 1, host : "192.168.100.85:27020" },
    { _id : 2, host : "192.168.100.85:27021" }
  ]
})
```

```
rs.initiate(
{
  _id: "master1rs",
  members: [
    { _id : 0, host : "192.168.100.85:27030" },
    { _id : 1, host : "192.168.100.85:27031" },
    { _id : 2, host : "192.168.100.85:27032" }
  ]
})
```

```
docker-compose -f config-server/docker-compose.yaml up -d
```

```
C:\Users\carli\OneDrive\Documentos\Universidad\6\Mod_Base_Datos\SegundoParcial\pro_V6>docker-compose -f docker-compose.yaml up -d
time="2023-07-12T22:42:30-05:00" level=warning msg="Found orphan containers ([configsvr4 mongorouter02]) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up."
[+] Running 13/13
✓ Container esc2a      Running
✓ Container mongorouter Running
✓ Container esc2b      Running
✓ Container escl1       Running
✓ Container master2     Running
✓ Container esc3a      Running
✓ Container esc3b      Running
✓ Container esc3c      Running
✓ Container esc3d      Running
✓ Container esc3e      Running
✓ Container esc3f      Running
✓ Container esc3g      Running
✓ Container esc3h      Running
✓ Container esc3i      Running
✓ Container esc3j      Running
```

```
test> rs.initiate(
...   {
...     _id: "configserver",
...     configsvr: true,
...     members: [
...       { _id : 0, host : "192.168.100.85:27019" },
...       { _id : 1, host : "192.168.100.85:27020" },
...       { _id : 2, host : "192.168.100.85:27021" }
...     ]
...   }
... )
{ ok: 1, lastCommittedOpTime: Timestamp({ t: 1689218564, i: 1 }) }
configserver [direct: other] test>
```

```
configserver [direct: primary] test> |
```

```
test> rs.initiate(
...   {
...     _id: "master1rs",
...     members: [
...       { _id : 0, host : "192.168.100.85:27030" },
...       { _id : 1, host : "192.168.100.85:27031" },
...       { _id : 2, host : "192.168.100.85:27032" }
...     ]
...   }
... )
{ ok: 1 }
master1rs [direct: other] test>
```

# CONFIGURACION:

```
sh.addShard("master1rs/192.168.100.85:27030,192.168.100.85:27031,192.168.100.85:27032")
sh.addShard("master2rs/192.168.100.85:27040,192.168.100.85:27041,192.168.100.85:27042")
```

```
mongosh mongodb://192.168.100.85:27017,192.168.100.85:27018,192.168.100.85:27019/test> sh.status()
{
  shardingVersion: {
    _id: 1,
    minCompatibleVersion: 5,
    currentVersion: 6,
    clusterId: ObjectId("64af6e0fe58aaa36ec3b2047")
  }
}
shards: [
  {
    _id: 'master1rs',
    host: 'master1rs/192.168.100.85:27030,192.168.100.85:27031,192.168.100.85:27032',
    state: 1,
    topologyTime: Timestamp({ t: 1689222765, i: 4 })
  },
  {
    _id: 'master2rs',
    host: 'master2rs/192.168.100.85:27040,192.168.100.85:27041,192.168.100.85:27042',
    state: 1,
    topologyTime: Timestamp({ t: 1689222777, i: 5 })
  }
]
```

## MONGO-ROUTER

Con el siguiente comando vamos a inicializar los sharding.

## CREAMOS LA BASE DE DATOS Y LA COLECCION:

```
[direct: mongos] test> use jrs_proyecto
switched to db jrs_proyecto
[direct: mongos] jrs_proyecto>
[direct: mongos] jrs_proyecto> db
jrs_proyecto
[direct: mongos] jrs_proyecto> db.createCollection("anime")
{ ok: 1 }
```

# CONFIGURACION:

Fragmentamos la colección en este caso es anime ( min 5:13 fragmentar tiene que ser partes completas no solo el nombre de la colección asi que las desmostraciones de fragmentos son el punto de la base de datos y el nombre de la colección y luego una clave de fragmento).

```
sh.enableSharding("jrs_proyecto")
```

```
[direct: mongos] jrs_proyecto> sh.shardCollection("jrs_proyecto.anime", {"anime_id": "hashed"})
{
  collectionsharded: 'jrs_proyecto.anime',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1689223467, i: 36 }),
    signature: {
      hash: Binary(Buffer.from("00000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1689223467, i: 32 })
}
[direct: mongos] jrs_proyecto> |
```

# API

Agregar Nuevo

Buscar por Anime ID

Buscar

ID	Anime ID	Nombre	Género	Tipo	Episodios	Rating	Members	Acciones
64b0573c2c07b8c13fe3b593	925329	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572	<button>Editar</button> <button>Eliminar</button>
64b0573c2c07b8c13fe3b594	9969	Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen	TV	51	9.16	151266	<button>Editar</button> <button>Eliminar</button>
64b0573c2c07b8c13fe3b595	32935	Haikyuu!!: Karasuno Koukou VS Shiratorizawa Gakuen Koukou	Comedy, Drama, School, Shounen, Sports	TV	10	9.15	93351	<button>Editar</button> <button>Eliminar</button>
64b0573c2c07b8c13fe3b596	11061	Hunter x Hunter (2011)	Action, Adventure, Shounen, Super Power	TV	148	9.13	425855	<button>Editar</button> <button>Eliminar</button>
64b0573c2c07b8c13fe3b597	820	Ginga Eiyuu Densetsu	Drama, Military, Sci-Fi, Space	OVA	110	9.11	80679	<button>Editar</button> <button>Eliminar</button>

Siguiente

Ver transformación

# QUE CONTIENE NUESTRA API



## BOTON EDITAR

Una vez seleccionamos la fila que deseamos editar se nos despliega un formulario, posteriormente editar el campo deseado.

Filter `{"anime_id":9253293}`

Reset Find Options ▾

0 - 0 of 0 < > { } [ ]

No results

Try modifying your query to get results.

### Dentro de mongo:

Filter `{"anime_id":9253293}`

Reset Find Options ▾

1 - 1 of 1 < > { } [ ]

```
_id: ObjectId('64b0573c2c07b8c13fe3b593')
anime_id: 9253293
name: "Steins;Gate"
genre: "Sci-Fi, Thriller"
type: "TV"
episodes: 24
rating: 9.17
members: 673572
```

Guardar

## Editar Anime

Anime ID:

9253293

Nombre:

Steins;Gate

Género:

Sci-Fi, Thriller

Tipo:

TV

Episodios:

24

Rating:

# QUE CONTIENE NUESTRA API

## BOTON AGREGAR

Agregar Nuevo

Se nos despliega un nuevo formulario, donde vamos ingresar los datos en cada uno de los campos.

Filter {"anime\_id":1722365143}

ADD DATA EXPORT DATA

**Dentro de mongo:**

```
_id: ObjectId('64b075b2df547389da26f88e')
anime_id: 1722365143
name: "Daniel Salazar"
genre: "masculino"
type: "TV"
episodes: 2
rating: 6
members: 1
__v: 0
```

## Agregar Nuevo Anime

Anime ID:

1722365143

Nombre:

Daniel Salazar

Género:

masculino

Tipo:

TV

Episodios:

2

Rating:

6

Members:

1

Guardar

# CONCLUSIONES

- En conclusión, la arquitectura de contenedores y ETL descrita ofrece una solución eficiente para la gestión de bases de datos y APIs en un entorno dockerizado. La estructura de contenedores permite una mayor escalabilidad, modularidad y disponibilidad del sistema, así como una gestión eficiente de los datos.
- Además, el uso de contenedores dockerizados ofrece ventajas en términos de portabilidad y despliegue de componentes del sistema. En resumen, esta arquitectura es una solución efectiva para la gestión de datos en entornos empresariales.

**GRACIAS  
POR SU  
ATENCION**

