

Instructivo de Instalación y Aplicación de GridFS para el manejo de imágenes

GridFS es el sistema de MongoDB para almacenar archivos grandes (>16MB) o cualquier archivo que quieras mantener en la base de datos en lugar del sistema de archivos. Divide los archivos en chunks y los almacena en dos colecciones: fs.files (metadatos) y fs.chunks (datos del archivo).

Para poder utilizar este sistema, se explicará el procedimiento que se utilizó paso a paso para su aplicación:

Creación de la Base de Datos en Mongo

1. Se crea el cluster invernadero en Mongo Atlas, utilizando la capa gratuita para efectos de ejercicio académico

Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10 \$0.08/hour

Dedicated cluster for development environments and low traffic applications.

STORAGE	RAM	VCPU
10 GB	2 GB	2 vCPUs

Flex From \$0.011/hour Up to \$30/month

For application development and testing, with on-demand burst capacity for unpredictable traffic.

STORAGE	RAM	VCPU
5 GB	Shared	Shared

Free

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	VCPU
512 MB	Shared	Shared

Free forever! Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Configurations

Name
You cannot change the name once the cluster is created.

Cluster name

Provider

aws Google Cloud Azure

Region

N. Virginia (us-east-1) ★ ★

★ Recommended ★ Low carbon emissions

Quick setup

☒ Automate security setup ⓘ

☐ Preload sample dataset ⓘ

Fill in this later

Go to Advanced Configuration

Create Deployment

2. Se exporta el string de conexión o Mongo_URI para poder acceder al cluster y a la base de datos mediante la aplicación Mongo Compass

Connect to invernadero



Connecting with MongoDB Compass

I don't have MongoDB Compass installed

I have MongoDB Compass installed

1. Choose your version of Compass

1.38 or later

See your Compass version in "About Compass"

2. Copy the connection string, then open MongoDB Compass

Use this connection string in your application

```
mongodb+srv://crushergeek12:<db_password>@invernadero.hz1bl0q.mongodb.net/
```

Replace `<db_password>` with the password for the `crushergeek12` user. Ensure any options are [URL encoded](#).
You can edit your database user password in [Database Access](#).

- Se introduce el string de conexión en el Mongo Compass, y le da al botón "Save and Connect" (Guardar y Conectarse), para conectarse al cluster invernadero

New Connection

Manage your connection settings

URI Edit Connection String

mongodb+srv://crushergeek12:*****@invernadero.hz1bl0q.mongodb.net/

Name invernadero.hz1bl0q.mongodb.net **Color** No Color

☐ **Favorite this connection**
Favoriting a connection will pin it to the top of your list of connections

Advanced Connection Options

General Authentication TLS/SSL Proxy/SSH In-Use Encryption Advanced

Connection String Scheme

mongodb mongodb+srv

DNS Seed List Connection Format. The uri indicates to the client that the hostname that follows

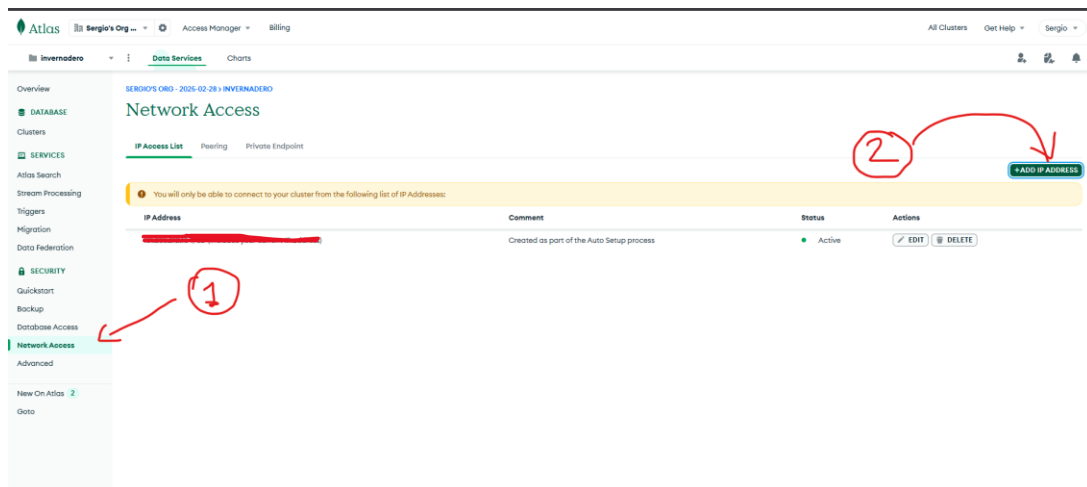
Cancel Save Connect **Save & Connect**

How do I find my connection string in Atlas?
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect. [See example](#)

How do I format my connection string?
[See example](#)

4. Para poder utilizar el cluster desde otros dispositivos y en otras redes, debemos ir en el Dashboard de Mongo Atlas al apartado “Network Access” de las opciones de seguridad, y agregar una nueva IP.

En el apartado de Add IP List, seleccionamos la opción “Allow Access From Anywhere” (Permitir acceso desde cualquier lugar), lo que agregará la IP 0.0.0.0/0, que facilitará distintas conexiones.



Add IP Access List Entry

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. [Learn more](#)

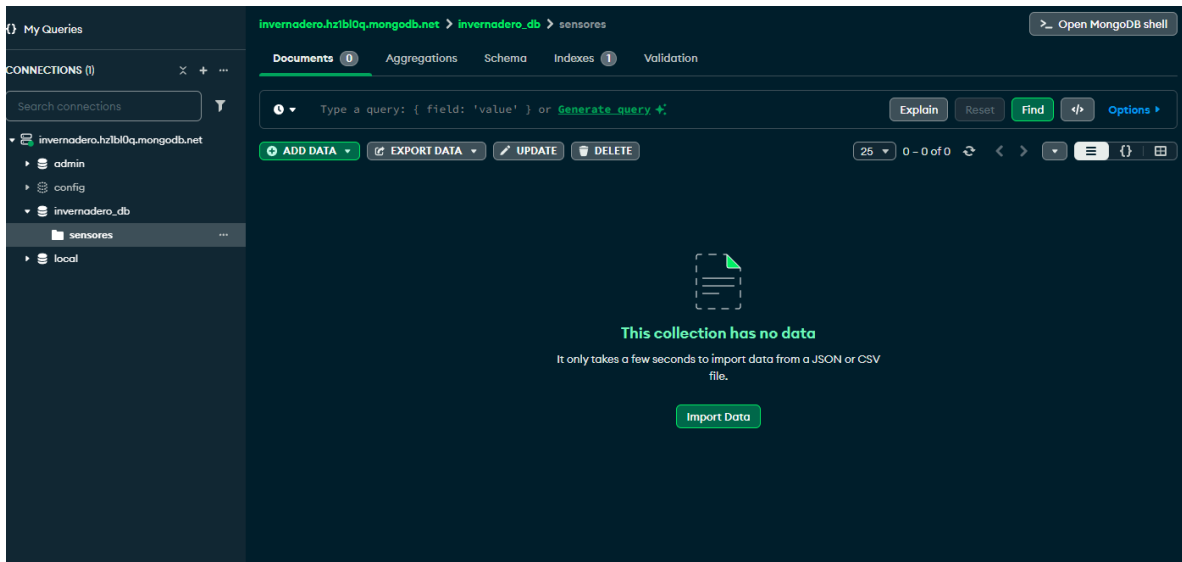
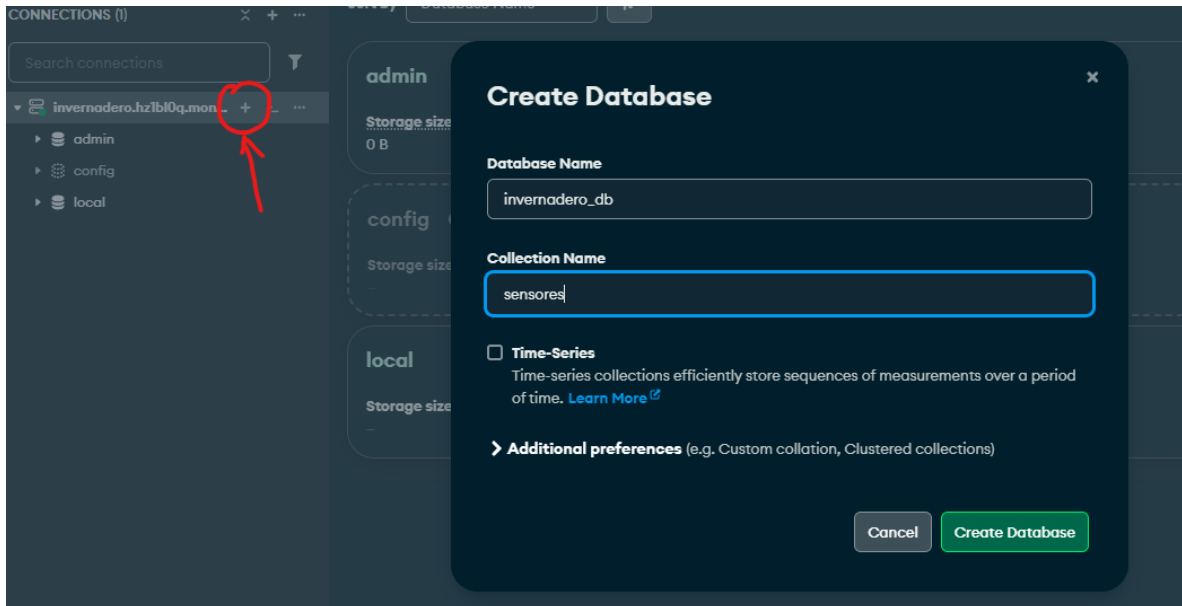
ALLOW ACCESS FROM ANYWHERE

Access List Entry:

Comment:

☐ This entry is temporary and will be deleted in 6 hours

5. En el Compass, creamos una nueva database llamada `invernadero_db` y con una colección de ejemplo llamada `sensores`



Preparación del entorno de desarrollo

1. Instalamos las dependencias necesarias para el backend (Enfásis en Pymongo, que es la que enlaza con Mongo, gridFs que es con lo que vamos a trabajar, y paho-mqtt para recibir la comunicación de los dispositivos)

```
pip install Flask pymongo gridfs Pillow ultralytics paho-mqtt python-dotenv
```

2. Identificamos la estructura de nuestro backend

```
proyecto/  
├── .env  
├── run.py  
├── app/  
│   ├── __init__.py  
│   ├── routes.py  
│   ├── db.py  
│   ├── ai_model.py  
│   └── mqtt_listener.py  
└── yolov8n.pt # Modelo YOLO
```

3. En el archivo .env, guardamos el enlace de conexión a nuestra base de datos

```
1 MONGO_URI= mongodb+srv://anguielaiseca123:Manzana123@invernadero.xfex8dr.mongodb.net/
```

4. En el archivo db.py, establecemos los nuestros parámetros de conexión con la base de datos

```
import os
from pymongo import MongoClient
from dotenv import load_dotenv

load_dotenv()

client = None

def init_db():
    global client
    MONGO_URI = os.getenv("MONGO_URI")
    client = MongoClient(MONGO_URI)

def get_db():
    global client

    if client is None:
        MONGO_URI = os.getenv("MONGO_URI")
        if not MONGO_URI:
            raise RuntimeError("❌ MONGO_URI no está definido en el archivo .env")
        client = MongoClient(MONGO_URI)

    return client["invernadero_db"]
```

Implementación de GridFS

1. Se importa gridFs, y se crea el método para guardar las imágenes, en el que recibe un JSON de la imagen y revisa el formato en el que llega (Se está trabajando con el formato base64), y se transforma en formato bytes.

```
@bp.route('/api/imagenes', methods=['POST'])
def guardar_imagen_esp32():
    try:
        data = request.get_json()

        # Validar que llegue la imagen en base64
        if not data or 'imagen_base64' not in data:
            return jsonify({"error": "El campo 'imagen_base64' es obligatorio"}), 400

        imagen_base64 = data['imagen_base64']

        # Decodificar base64 a bytes
        try:
            # Remover prefijo si existe (data:image/jpeg;base64,)
            if ',' in imagen_base64:
                imagen_base64 = imagen_base64.split(',')[1]

            imagen_bytes = base64.b64decode(imagen_base64)
        except Exception as e:
            return jsonify({"error": f"Error decodificando base64: {str(e)}"}), 400

        # Validar que sea una imagen válida
        try:
            imagen_pil = Image.open(io.BytesIO(imagen_bytes))
            imagen_pil.verify() # Verifica que sea una imagen válida
        except Exception as e:
            return jsonify({"error": f"Archivo no es una imagen válida: {str(e)}"}), 400
```

- Posteriormente, se crea un objeto fs de gridFS, que se utilizará para gestionar archivos binarios (Guardado y Recuperación de Imágenes). Se obtiene el metadata de la imagen (Hora de creación, Fuente, Tipo de Contenido), y el objeto fs se encargará de almacenar la imagen en MongoDB, generando un file_id.

```
# Guardar en GridFS
db = get_db()
fs = GridFS(db)

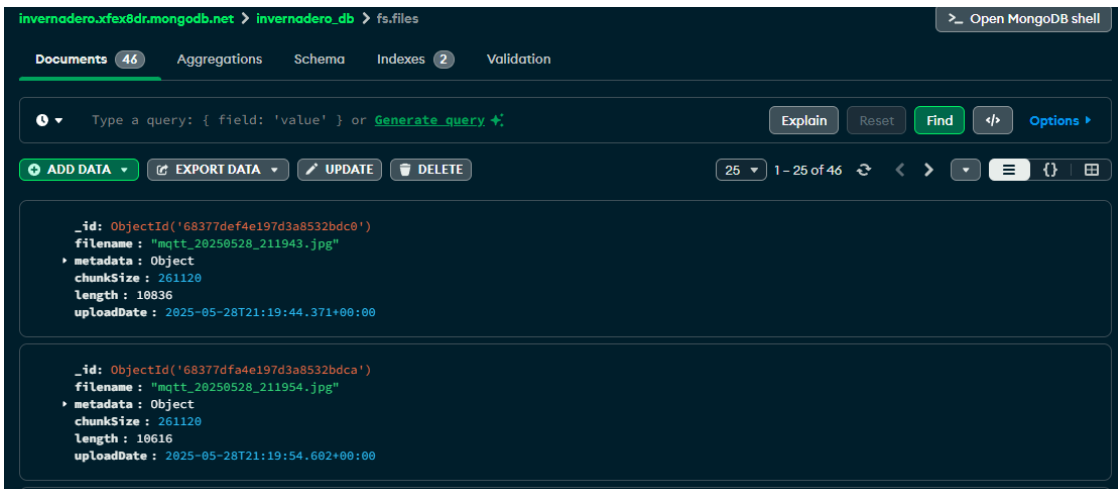
# Metadatos básicos
metadata = {
    'timestamp': datetime.utcnow(),
    'source': 'esp32',
    'content_type': 'image/jpeg'
}

# Guardar imagen en GridFS
file_id = fs.put(
    imagen_bytes,
    filename=f"esp32_{datetime.utcnow().strftime('%Y%m%d_%H%M%S')}.jpg",
    metadata=metadata
)

return jsonify({
    "mensaje": "Imagen guardada exitosamente",
    "file_id": str(file_id)
}), 201

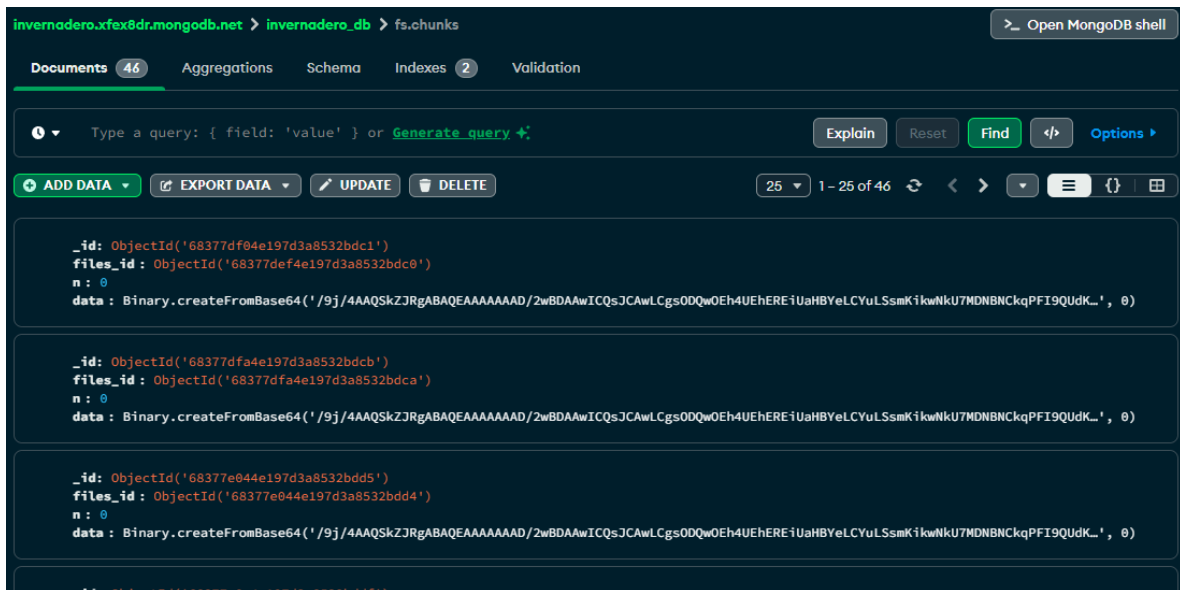
except Exception as e:
    return jsonify({"error": f"Error interno del servidor: {str(e)}"}), 500
```

- En MongoDB, al utilizar gridFS, se crearán dos carpetas automáticamente: fs.file (metadata) y fs.chunks (datos del archivo), que se referencian con el file_id.



The screenshot shows the MongoDB Compass interface for the 'invernadero' database, specifically the 'fs.files' collection. The interface includes tabs for Documents, Aggregations, Schema, Indexes, and Validation. A query bar at the top allows for searching documents. Below the query bar, there are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The main area displays two document entries. Each entry shows the '_id' as an ObjectId, the 'filename' as 'mqtt_20250528_211943.jpg' and 'mqtt_20250528_211954.jpg' respectively, and the 'metadata' as an object containing 'chunkSize', 'length', and 'uploadDate'.

Document	_id	filename	metadata
1	ObjectId('68377def4e197d3a8532bdc0')	"mqtt_20250528_211943.jpg"	{ "chunkSize": 261120, "length": 16836, "uploadDate": "2025-05-28T21:19:44.371+00:00" }
2	ObjectId('68377dfa4e197d3a8532bdca')	"mqtt_20250528_211954.jpg"	{ "chunkSize": 261120, "length": 16616, "uploadDate": "2025-05-28T21:19:54.602+00:00" }



4. Se recupera la imagen de la base de datos mediante el file_id que enlaza los chunks y los files, y se entrega el resultado de la imagen como .jpg.

```
@bp.route('/api/imagenes/<string:file_id>', methods=['GET'])
def obtener_imagen_gridfs(file_id):
    try:
        db = get_db()
        fs = GridFS(db)

        file = fs.get(ObjectId(file_id))

        return send_file(
            BytesIO(file.read()),
            mimetype='image/jpeg',
            as_attachment=False,
            download_name=f"imagen_{file_id}.jpg"
        )

    except Exception as e:
        return jsonify({"error": f"Error obteniendo imagen: {str(e)}"}), 404
```

Implementaciones del Sistema

Para integrar el almacenamiento y recuperación de imágenes con gridFS, con otras aplicaciones del sistema (Fotos enviada por microcontroladores, Predicción de Objetos en Imágenes), se debe hacer lo siguiente:

1. En el archivo mqtt_listener.py, importamos mqtt de la librería paho-mqtt para configurar el bróker mqtt para la comunicación con los microdispositivos, llamando al tópicos encargado de tomar las fotos (camara/foto)

```
# === CONFIGURACIÓN DEL BROKER MQTT ===
MQTT_BROKER = "192.168.204.153"
MQTT_PORT = 1883

# === TOPICS ===
TOPIC_AMBIENTE = "iot/ambiente"
TOPIC_NIVEL = "iot/nivel"
TOPIC_IMAGEN = "camara/foto"
TOPIC_DETECCION = "deteccion/personas"
TOPIC_TEMP_PRED = "iot/temperatura/prediccion"
TOPIC_HUM_PRED = "iot/humedad/prediccion"
```

2. Se recibe el mensaje que envía el microdispositivo con el tópicos "TOPIC_IMAGEN" (Foto tomada en formato base64) y se guarda usando gridFS en la base de datos

```
elif msg.topic == TOPIC_IMAGEN:
    imagen_base64 = msg.payload.decode()

    if ',' in imagen_base64:
        imagen_base64 = imagen_base64.split(',')[1]

    imagen_bytes = base64.b64decode(imagen_base64)
    imagen_pil = Image.open(io.BytesIO(imagen_bytes))
    imagen_pil.verify()

    fs = GridFS(db)
    metadata = {
        'timestamp': datetime.utcnow(),
        'source': 'mqtt_esp32',
        'content_type': 'image/jpeg'
    }

    file_id = fs.put(
        imagen_bytes,
        filename=f"mqtt_{datetime.utcnow().strftime('%Y%m%d_%H%M%S')}.jpg",
        metadata=metadata
    )

    print(f"[✓] Imagen guardada desde MQTT con ID: {file_id}")
```

3. Del modelo escogido para el proyecto(YOLOv8), se llama la función de detectar_objetos, y se aplica en el mqtt_listener.py a la imagen que se guarda en Mongo con GridFS, para detectar objetos (Ej. Detectar cuantas personas hay en la foto)

```
def detectar_objetos(imagen_np):  
    imagen_bgr = cv2.cvtColor(imagen_np, cv2.COLOR_RGB2BGR)  
    resultados = modelo_yolo(imagen_bgr)  
  
    objetos = []  
    for r in resultados:  
        nombres = r.names  
        clases_detectadas = r.boxes.cls.tolist()  
        objetos = [nombres[int(clase)] for clase in clases_detectadas]  
  
    return objetos
```

```
print(f"[✓] Imagen guardada desde MQTT con ID: {file_id}")  
  
imagen_pil = Image.open(io.BytesIO(imagen_bytes)).convert("RGB").resize((640, 640))  
imagen_np = np.array(imagen_pil)  
  
objetos = detectar_objetos(imagen_np)  YeisonAndresDiazPerdomo, 4 hours ago • No copiarse  
  
resultado = {  
    "mensaje": "Detección automática desde MQTT",  
    "objetos_detectados": objetos,  
    "file_id": str(file_id),  
    "fecha_deteccion": datetime.utcnow().isoformat()  
}  
  
publicar_a_mqtt(resultado, TOPIC_DETECCION)  
print(f"[👁️] Objetos detectados automáticamente: {objetos}")  
  
else:  
    print(f"[⚠️] Topic no manejado: {msg.topic}")  
  
except Exception as e:  
    print(f"[X] Error procesando mensaje MQTT ({msg.topic}): {e}")
```

Flujo general de uso

1. Cliente envía imagen codificada en base64 en JSON al endpoint /guardar-imagenes
2. Backend decodifica la imagen y la guarda en GridFS.
3. Se devuelve al cliente el ID de la imagen guardada.
4. Para recuperar la imagen, el cliente consulta /obtener-imagenes/<image_id>.
5. El backend obtiene la imagen de GridFS y la envía para visualización o descarga.