1. **¿Cuál es el propósito de `module.exports`?**

- **Respuesta**:
  `module.exports` es un objeto en Node.js que se utiliza para exportar funciones, objetos o valores desde un módulo (archivo) para que puedan ser utilizados en otros módulos. Es la forma de hacer que ciertas partes del código estén disponibles para otros archivos.
  Ejemplo:

```
// math.js
module.exports = {
  sum: (a, b) => a + b,
  subtract: (a, b) => a - b,
};

// app.js
const math = require('./math');
console.log(math.sum(2, 3)); // 5
```

2. **¿Qué es un middleware?**

- **Respuesta**:
  Un middleware es una función que se ejecuta en el ciclo de solicitud-respuesta de una aplicación web. Tiene acceso al objeto de solicitud (`req`), al objeto de respuesta (`res`) y a la siguiente función middleware en la cadena (`next`). Se utiliza para realizar tareas como validación, autenticación, logging, etc.
  Ejemplo en Express.js:

```
app.use((req, res, next) => {
  console.log('Middleware ejecutado');
  next(); // Pasa al siguiente middleware o ruta
});
```

3. **¿Cuál es la diferencia entre código bloqueante y código no bloqueante?**

- **Respuesta**:
  - **Código bloqueante**: Es código que detiene la ejecución del programa hasta que se completa una operación (por ejemplo, leer un archivo o

hacer una consulta a la base de datos). Esto puede causar que la aplicación se congele mientras espera.

- **Código no bloqueante**: Es código que permite que la aplicación continúe ejecutándose mientras se realiza una operación asíncrona. Cuando la operación termina, se ejecuta un callback o una promesa.
  Ejemplo:

```
// Bloqueante (sincrónico)
const data = fs.readFileSync('file.txt'); // El programa se detiene aquí
console.log(data);

// No bloqueante (asincrónico)
fs.readFile('file.txt', (err, data) => {
  if (err) throw err;
  console.log(data); // Se ejecuta cuando el archivo se ha leído
});
```

4. **¿Qué biblioteca de JavaScript usaría para manejar datos en tiempo real?**

- **Respuesta**:
  Para manejar datos en tiempo real, una de las bibliotecas más populares es **Socket.IO**. Permite la comunicación bidireccional en tiempo real entre el cliente y el servidor, ideal para aplicaciones como chats, notificaciones en tiempo real o juegos multijugador.

Ejemplo:

```
// Servidor
const io = require('socket.io')(3000);
io.on('connection', (socket) => {
  console.log('Usuario conectado');
  socket.on('message', (data) => {
    io.emit('message', data); // Envía el mensaje a todos los clientes
  });
```

```
});

// Cliente
const socket = io('http://localhost:3000');
socket.emit('message', 'Hola mundo');
```

Preguntas PLUS sobre Docker y Kubernetes

1. **¿Cuál es la principal ventaja de trabajar un proyecto dockerizado?**

- **Respuesta**:
  La principal ventaja de dockerizar un proyecto es la **portabilidad y consistencia**. Docker permite empaquetar una aplicación con todas sus dependencias en un contenedor, lo que garantiza que funcione de la misma manera en cualquier entorno (desarrollo, pruebas, producción). Además, facilita la escalabilidad y el despliegue.

2. **¿Cuál es la diferencia entre una imagen y un volumen en Docker?**

- **Respuesta**:
  - **Imagen**: Es una plantilla de solo lectura que contiene el código, las dependencias y el entorno necesario para ejecutar una aplicación. Se utiliza para crear contenedores.
  - **Volumen**: Es un mecanismo para persistir datos generados por los contenedores. Los volúmenes permiten almacenar datos fuera del contenedor, lo que evita que se pierdan cuando el contenedor se detiene o se elimina.

Ejemplo:

```
# Crear un volumen
docker volume create my_volume

# Usar un volumen en un contenedor
docker run -v my_volume:/app/data my_image
```

3. **¿Con qué herramienta se puede orquestar un proyecto con múltiples imágenes en Docker?**

- **Respuesta**:
  La herramienta más utilizada para orquestar proyectos con múltiples imágenes en Docker es **Docker Compose**. Permite definir y gestionar aplicaciones multi-contenedor en un solo archivo YAML.
  Ejemplo de `docker-compose.yml`:

```yaml
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example
```

4. **¿Cuál es la principal ventaja de trabajar con un clúster de Kubernetes?**

- **Respuesta**:
  La principal ventaja de Kubernetes es la **automatización y escalabilidad** en la gestión de aplicaciones en contenedores. Kubernetes permite:
  - Escalar automáticamente las aplicaciones según la demanda.
  - Gestionar el despliegue, actualización y rollback de aplicaciones.
  - Distribuir la carga entre múltiples nodos.
  - Asegurar alta disponibilidad y tolerancia a fallos.