



# Universidad ORT Uruguay

## Facultad de Ingeniería

# Arquitectura de Software en la Práctica

## Obligatorio 1

### Profesores

Guillermo Areosa  
Daniel Paggiola

### Estudiantes

Santiago Díaz - 240926

### Links de interes

Colección de Endpoints (swagger):

<http://feature-toggle-env-2.eba-pnu7hm3g.us-east-1.elasticbeanstalk.com/api-docs/>

Repositorio:

<https://github.com/SantiagoDiaz9822/ASP-obligatorio>

Dominio Elastic Beanstalk:

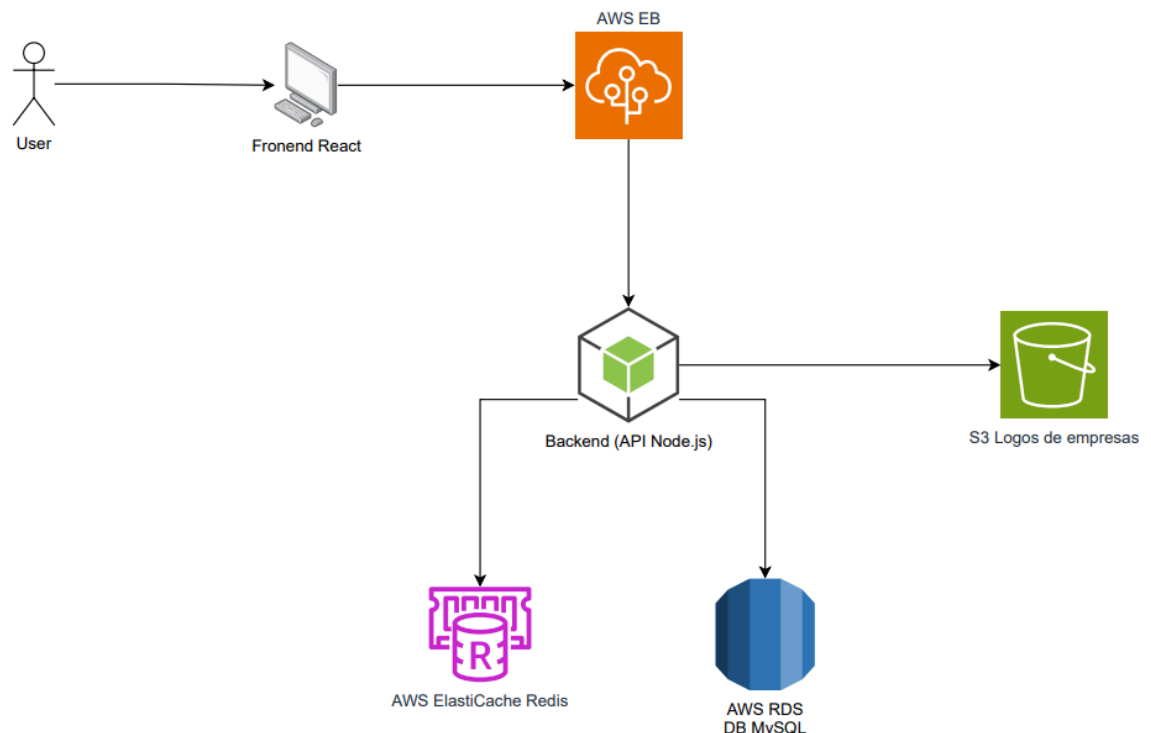
<http://feature-toggle-env-2.eba-pnu7hm3g.us-east-1.elasticbeanstalk.com>

<b>1. Descripción de la arquitectura</b>	<b>2</b>
Estructuras:	2
Distribución física	2
Ciclo de vida de una petición:	3
<b>2. Credenciales para el uso de la aplicación</b>	<b>3</b>
<b>3. Justificación de Diseño para los Requerimientos No Funcionales</b>	<b>4</b>
RNF1. Performance	4
RNF2. Confiabilidad y disponibilidad	4
RNF3. Observabilidad	5
RNF4. Autenticación, autorización y tenancy	5
RNF5. Seguridad	6
RNF6. Código fuente	6
RNF7. Integración continua	6
RNF8. Pruebas de carga	9
RNF9. Identificación de fallas	10
RNF10. Portabilidad	11
<b>4. Cumplimiento con los 12 factores de una app</b>	<b>11</b>
<b>5. Descripción del proceso de Deployment</b>	<b>13</b>
Plataforma	13
Proceso de Producción	14

# 1. Descripción de la arquitectura

## Estructuras:

1. Backend:
  - Node.js: Servidor que maneja las peticiones y responde a las solicitudes del frontend.
2. Frontend:
  - React: Aplicación que interactúa con el backend mediante peticiones HTTP.
3. AWS:
  - Elastic Beanstalk: Plataforma para desplegar y administrar el backend.
  - RDS (Relational Database Service): Servicio para gestionar la base de datos MySQL.
  - ElastiCache: Servicio de caché en memoria para Redis.
  - S3: Bucket para guardar y consultar los logos de las empresas creadas.



## Distribución física

- Servidor Backend: Despliegue en AWS Elastic Beanstalk.
- Base de datos: Amazon RDS para MySQL, donde se almacenan los datos de la aplicación.
- Caché: Amazon ElastiCache para Redis, donde se almacenan datos en caché para mejorar el rendimiento.
- Frontend: Aplicación React actualmente no se despliega.

## Ciclo de vida de una petición:

1. Petición del usuario: Un usuario interactúa con la aplicación React.
2. Solicitud HTTP: La aplicación realiza una solicitud HTTP al dominio del Elastic Beanstalk.
3. Verificación de autenticación: El backend utiliza middleware para verificar el token JWT del usuario.
4. Consulta de caché: El backend consulta Redis para obtener datos almacenados en caché:
  - a. Si los datos están en caché, se envían de regreso al frontend.
  - b. Si no están, el backend realiza una consulta a la base de datos MySQL.
5. Almacenamiento en caché: Si se obtienen nuevos datos de MySQL, estos se almacenan en Redis para consultas futuras.
6. Respuesta: El backend envía la respuesta al frontend, que muestra los datos al usuario.

## 2. Credenciales para el uso de la aplicación

Usuario administrador:

- Email: admin@ort.com
- Contraseña: Password123!

Usuario común:

- Email: usuario@ort.com
- Contraseña: 123456789

### 3. Justificación de Diseño para los Requerimientos No Funcionales

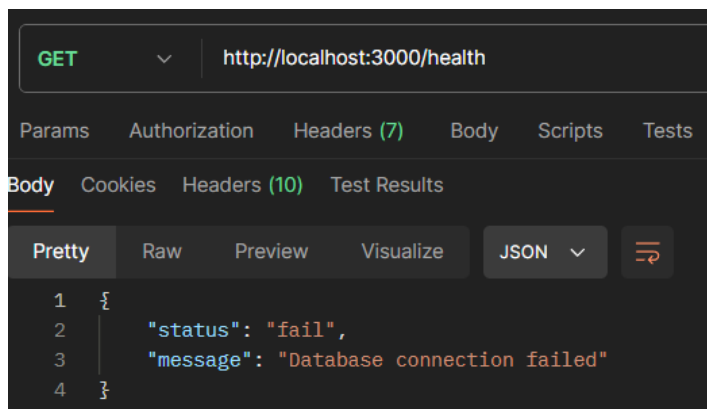
#### RNF1. Performance

- **Diseño aplicado:** Para cumplir con el requisito de responder las consultas del RF6 en menos de 100 ms para cargas de hasta 1200 req/m, se implementó un sistema de caching utilizando Redis (Amazon ElastiCache). Redis almacena temporalmente los resultados de consultas que requieren mucho procesamiento, lo que reduce la carga en la base de datos y mejora significativamente los tiempos de respuesta.
- **Tácticas aplicadas:** Redis se utiliza como una capa de almacenamiento rápido en memoria. Además, la estructura de la base de datos ha sido optimizada con índices en los campos de búsqueda clave. Finalmente, se han realizado ajustes de configuración en AWS para asegurar que la infraestructura pueda manejar el tráfico previsto.

#### RNF2. Confiabilidad y disponibilidad

- **Diseño aplicado:** Se implementó un endpoint HTTP de acceso público `/health` que verifica la conectividad con la base de datos MySQL, Redis y otros componentes clave. Este endpoint permite el monitoreo de la disponibilidad del sistema.
- **Tácticas aplicadas:** Este endpoint hace una llamada simple a los servicios principales y retorna un estado HTTP 200 si todo funciona correctamente, o un 500 si hay algún problema.

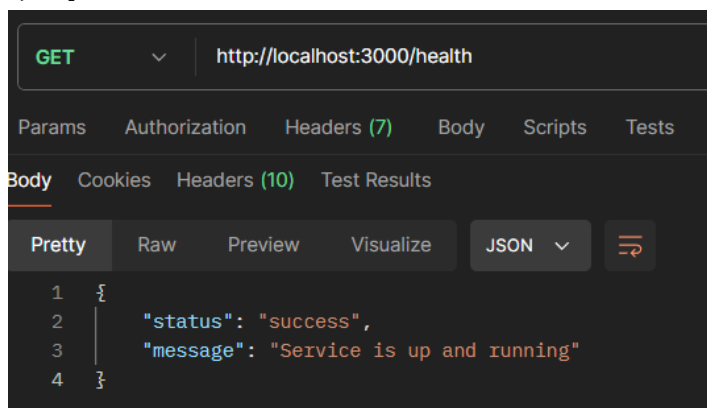
Ejemplo base de datos no corriendo:



A screenshot of a REST client interface. The top bar shows a GET request to `http://localhost:3000/health`. Below the bar, the 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response is a failure:

```
1 {
2   "status": "fail",
3   "message": "Database connection failed"
4 }
```

Ejemplo sistema corriendo correctamente:



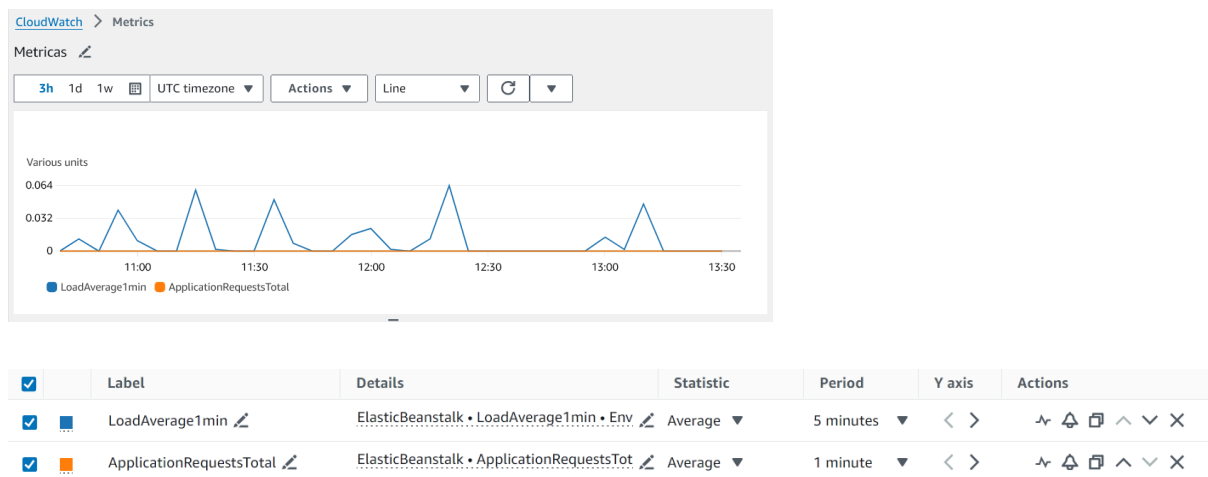
A screenshot of a REST client interface, similar to the one above. The top bar shows a GET request to `http://localhost:3000/health`. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response is a success:

```
1 {
2   "status": "success",
3   "message": "Service is up and running"
4 }
```

### RNF3. Observabilidad

- **Diseño aplicado:** Para monitorear el rendimiento y el uso del sistema, se integro la herramienta de monitoreo AWS CloudWatch que recolecta métricas de rendimiento como el número de peticiones por minuto y los tiempos de respuesta a un endpoint.
- **Tácticas aplicadas:** Se configuraron logs detallados para capturar métricas de las peticiones y tiempos de respuesta en cada endpoint del backend. Estas métricas se envían a una herramienta de monitoreo que las procesa y almacena.

Metricas CloudWatch:



### RNF4. Autenticación, autorización y tenancy

- **Diseño aplicado:** El sistema implementa control de acceso basado en roles, utilizando JSON Web Tokens (JWT) para la autenticación y bcrypt para hashear las contraseñas de los usuarios, asegurando que la información sensible esté protegida. Además, se utiliza middleware personalizado para proteger las rutas y autorizar el acceso a cada una de las funcionalidades según los roles de los usuarios.
- **Tácticas aplicadas:** Las rutas del backend están protegidas por un middleware que verifica si el token JWT del usuario es válido y tiene los permisos adecuados para acceder al recurso solicitado. Este middleware también asegura que los usuarios solo puedan acceder a los datos de su propia empresa. El uso de bcrypt para el hashing de contraseñas garantiza que estas se almacenen de forma segura y no sean recuperables en caso de acceso no autorizado a la base de datos.

Ejemplo de contraseña hasheada:

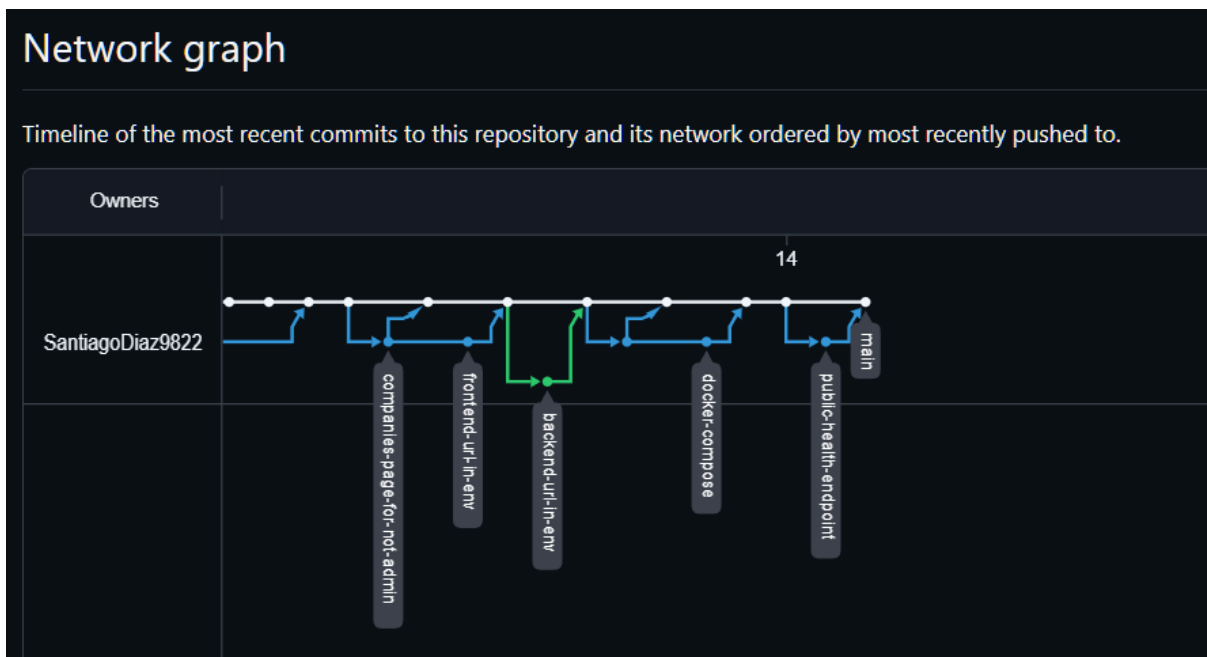
```
{
  "id": 5,
  "company_id": 6,
  "email": "userupdated@example.com",
  "password_hash": "$2b$10$ax2kItELVcM5a4Uq49kV2.vc9duREg1f0RfxAzSZJnTd1lT6phtpm",
  "role": "admin",
  "first_login": 1,
  "created_at": "2024-10-10T15:45:04.000Z",
  "updated_at": "2024-10-12T22:26:52.000Z"
},
```

## RNF5. Seguridad

- **Diseño aplicado:** Actualmente, el frontend se comunica con el backend desplegado en Elastic Beanstalk a través de HTTP utilizando el dominio. Esto implica que las comunicaciones no están cifradas, lo que representa un riesgo de seguridad.
- **Tácticas aplicadas:** Para cumplir con este requerimiento en producción, se planea configurar HTTPS en Elastic Beanstalk. Esto se logrará adquiriendo y configurando un certificado SSL/TLS para habilitar la comunicación segura. El uso de HTTPS garantizará que las transmisiones de datos entre el frontend y el backend estén protegidas frente a interceptaciones.

## RNF6. Código fuente

- **Diseño aplicado:** El proyecto se gestiona con GitHub, siguiendo la metodología GitHub Flow. El archivo [README.md](#) incluye instrucciones detalladas sobre cómo configurar el ambiente de desarrollo y las convenciones de manejo de branches.
- **Tácticas aplicadas:** El uso de Pull Requests y revisiones de código asegura la calidad del código. La metodología GitHub Flow permite un ciclo de vida de desarrollo simplificado y una integración continua ágil.



## RNF7. Integración continua

- **Diseño aplicado:** Se configuró un pipeline de integración continua (CI) que ejecuta pruebas unitarias automáticas para los requerimientos funcionales RF3. Definición de Proyectos y RF7. Reporte de Uso, asegurando una cobertura del 100%. Las pruebas se ejecutan automáticamente cada vez que se integra un commit a la rama principal.
- **Tácticas aplicadas:** Para esto, se configuró la herramienta de CI/CD GitHub Actions, que ejecuta pruebas de cada funcionalidad crítica del sistema en cada commit. Esto garantiza que las funcionalidades se mantengan estables durante el desarrollo.

Cobertura:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	93.47	91.66	90	93.47	
middleware	35.71	0	0	35.71	8-20
auth.js	35.71	0	0	35.71	
routes	100	100	100	100	
projects.js	100	100	100	100	
usageLogs.js	100	100	100	100	



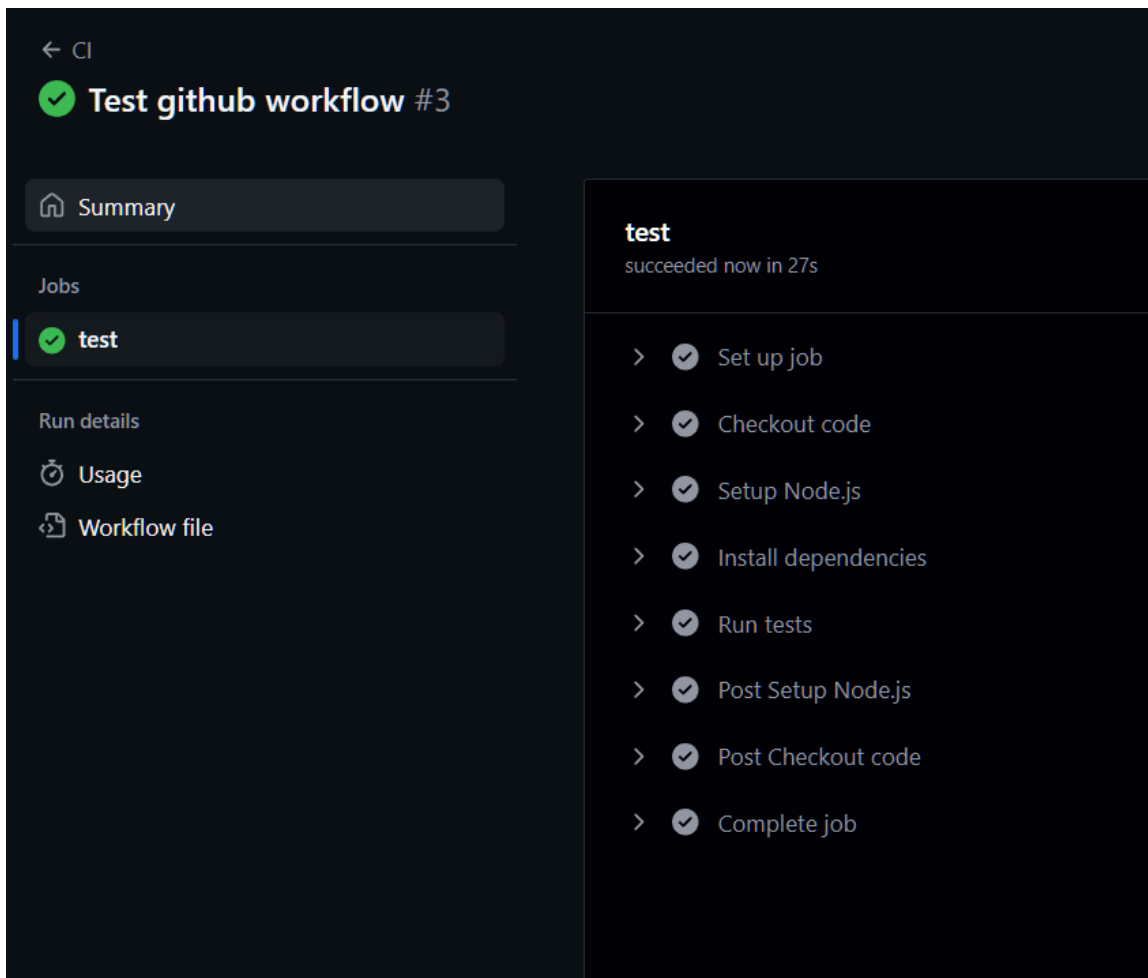
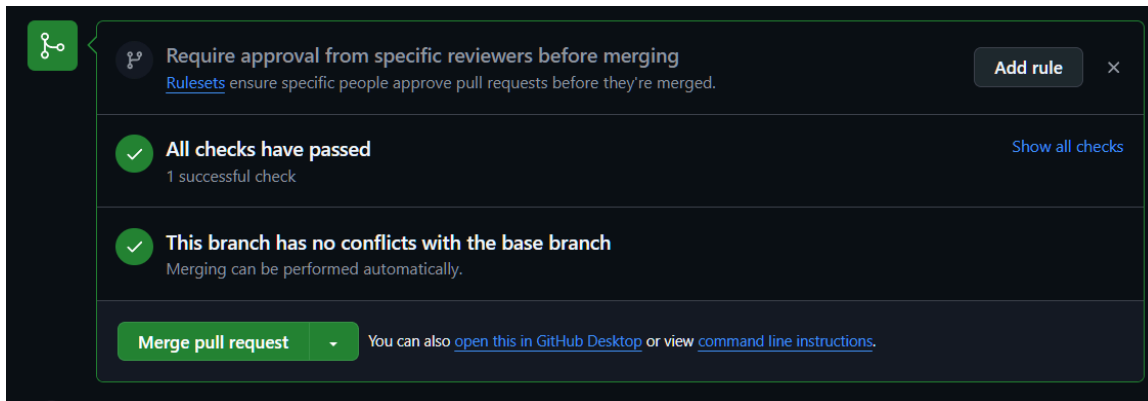
Workflow file:

```
Workflow file for this run
.github/workflows/main.yml at 374d3ab

1  name: CI
2
3  on:
4    push:
5      branches:
6        - main
7    pull_request:
8      branches:
9        - main
10
11 jobs:
12   test:
13     runs-on: ubuntu-latest
14
15     steps:
16     - name: Checkout code
17       uses: actions/checkout@v2
18
19     - name: Setup Node.js
20       uses: actions/setup-node@v2
21       with:
22         node-version: '14'
23
24     - name: Install dependencies
25       working-directory: ./backend
26       run: npm install
27
28     - name: Run tests
29       working-directory: ./backend
30       run: npm test
```

Ejemplo de uso con pull request:

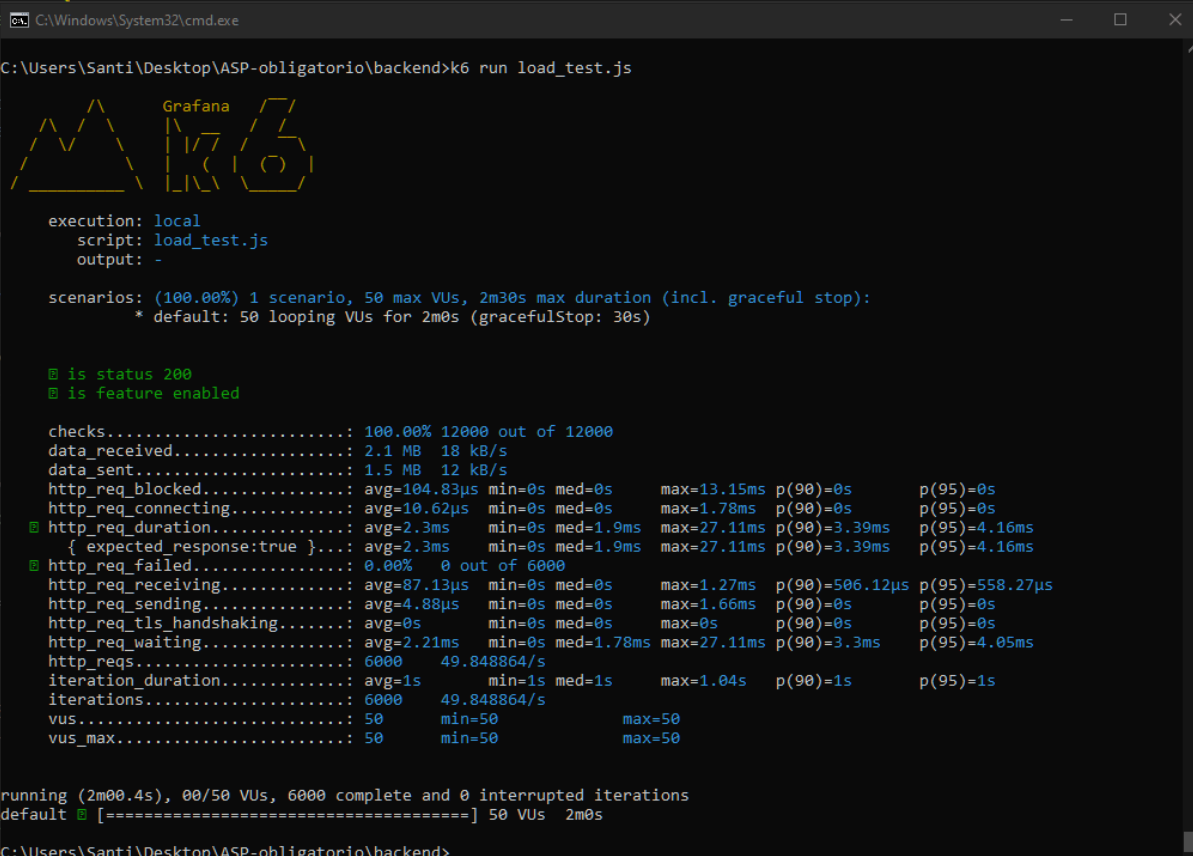
The screenshot shows a GitHub pull request interface. At the top, there's a summary bar with a yellow circle icon and the text "Some checks haven't completed yet" and "1 in progress check". To the right of this bar is a "Hide all checks" link. Below the summary bar, there are two check items. The first item has a yellow circle icon with a play button and the text "CI / test (pull\_request) In progress — This check has started...". To the right of this item is a "Details" link. The second item has a green circle icon with a checkmark and the text "This branch has no conflicts with the base branch". Below this item, it says "Merging can be performed automatically." At the bottom of the interface, there is a "Merge pull request" button with a dropdown arrow, followed by the text "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)."



## RNF8. Pruebas de carga

- **Diseño aplicado:** Se realizaron pruebas de carga utilizando la herramienta k6.io, simulando el comportamiento de la aplicación bajo distintas cargas de usuarios para asegurar que pueda manejar las 1200 req/m con un tiempo de respuesta menor a 100 ms.
- **Tácticas aplicadas:** Se crearon scripts que generan múltiples solicitudes simultáneas a los endpoints críticos, evaluando el comportamiento del sistema bajo condiciones de alta demanda y ajustando parámetros de la infraestructura si fuera necesario.

Evidencia prueba de carga:



```
C:\Windows\System32\cmd.exe
C:\Users\Santi\Desktop\ASP-obligatorio\backend>k6 run load_test.js

  execution: local
    script: load_test.js
    output: -

  scenarios: (100.00%) 1 scenario, 50 max VUs, 2m30s max duration (incl. graceful stop):
    * default: 50 looping VUs for 2m0s (gracefulStop: 30s)

  is status 200
  is feature enabled

  checks.....: 100.00% 12000 out of 12000
  data_received.....: 2.1 MB 18 kB/s
  data_sent.....: 1.5 MB 12 kB/s
  http_req_blocked.....: avg=104.83µs min=0s med=0s max=13.15ms p(90)=0s p(95)=0s
  http_req_connecting.....: avg=10.62µs min=0s med=0s max=1.78ms p(90)=0s p(95)=0s
  http_req_duration.....: avg=2.3ms min=0s med=1.9ms max=27.11ms p(90)=3.39ms p(95)=4.16ms
  { expected_response:true }...: avg=2.3ms min=0s med=1.9ms max=27.11ms p(90)=3.39ms p(95)=4.16ms
  http_req_failed.....: 0.00% 0 out of 6000
  http_req_receiving.....: avg=87.13µs min=0s med=0s max=1.27ms p(90)=506.12µs p(95)=558.27µs
  http_req_sending.....: avg=4.88µs min=0s med=0s max=1.66ms p(90)=0s p(95)=0s
  http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
  http_req_waiting.....: avg=2.21ms min=0s med=1.78ms max=27.11ms p(90)=3.3ms p(95)=4.05ms
  http_reqs.....: 6000 49.848864/s
  iteration_duration.....: avg=1s min=1s med=1s max=1.04s p(90)=1s p(95)=1s
  iterations.....: 6000 49.848864/s
  vus.....: 50 min=50 max=50
  vus_max.....: 50 min=50 max=50

  running (2m00.4s), 00/50 VUs, 6000 complete and 0 interrupted iterations
  default [=====] 50 VUs 2m0s

C:\Users\Santi\Desktop\ASP-obligatorio\backend>
```

Resultados:

- Total de solicitudes: 6000
- Tasa de éxito: 100%
- Tiempo promedio de respuesta: 2.3 ms
- Tiempos de respuesta máximos: 27.11 ms
- Porcentaje de solicitudes por debajo de 200 ms: 100%
- Tasa de fallos: 0%

Los resultados demuestran que el sistema es capaz de manejar la carga esperada sin problemas significativos de rendimiento.

## RNF9. Identificación de fallas

- **Diseño aplicado:** Los logs de la aplicación en producción se centralizan utilizando AWS CloudWatch Logs, reteniendo estos logs durante un período de al menos 24 horas. Estos logs ayudan a identificar fallas de manera eficiente.
- **Tácticas aplicadas:** Cada servicio genera logs detallados de errores, accesos, y fallos en un formato que permite su análisis posterior. Los logs se almacenan y se pueden consultar a través de una interfaz gráfica para diagnóstico en caso de fallas.

CloudWatch > Log groups

Log groups (13)  
By default, we only load up to 10000 log groups.



Filter log groups or try prefix search  ☐ Exact match

< 1 > ⚙

<input type="checkbox"/> Log group	Log class	Anomaly d...	Data protection	Sensitive data count	Retention
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/docker</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/docker-compose-events.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/docker-events.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/eb-docker/containers/eb-current-app/stdouterr.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/eb-docker/containers/eb-current-app/unexpected-quit.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/eb-engine.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/eb-hooks.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/nginx/access.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-1/var/log/nginx/error.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/elasticbeanstalk/Feature-toggle-env-2/environment-health.log</a>	Standard	<a href="#">Configure</a>	-	-	1 week
<input type="checkbox"/> <a href="#">/aws/lambda/RedshiftEventSubscription</a>	Standard	<a href="#">Configure</a>	-	-	Never expire
<input type="checkbox"/> <a href="#">/aws/lambda/RedshiftOverwatch</a>	Standard	<a href="#">Configure</a>	-	-	Never expire
<input type="checkbox"/> <a href="#">/aws/lambda/RoleCreationFunction</a>	Standard	<a href="#">Configure</a>	-	-	Never expire

## RNF10. Portabilidad

- **Diseño aplicado:** Se proporcionan archivos de configuración para Docker que permiten que cualquier persona que clone el repositorio pueda levantar el sistema localmente usando un solo comando (**docker-compose up**).
- **Tácticas aplicadas:** El uso de contenedores Docker asegura que la aplicación se ejecute de manera consistente en cualquier entorno, y el archivo **docker-compose.yml** facilita la configuración automática de todos los servicios necesarios (base de datos, backend, frontend, Redis).

<input type="checkbox"/>		<b>asp-obligatorio</b>	Running (3/3)	0.48%	3.47%	2	■	:	🗑
<input type="checkbox"/>		<b>frontend-1</b> f55f6233437e	<a href="#">asp-obligatorio-fro</a> Running	0.02%	3.08%	<a href="#">3001:3000</a>	2	■	:
<input type="checkbox"/>		<b>backend-1</b> 9d58a66b7d5f	<a href="#">asp-obligatorio-ba</a> Running	0.03%	0.36%	<a href="#">3000:3000</a>	2	■	:
<input type="checkbox"/>		<b>redis-1</b> 0665d5175f07	<a href="#">redis:latest</a> Running	0.43%	0.03%	<a href="#">6379:6379</a>	2	■	:

## 4. Cumplimiento con los 12 factores de una app

### 1. Codebase:

- Se utiliza un único repositorio que mantiene tanto el frontend como el backend de la aplicación. Esto facilita la gestión del código y asegura que ambas partes del sistema estén sincronizadas.

### 2. Config:

- La configuración de la aplicación se gestiona a través de variables de entorno, utilizando un archivo .env. Esto permite cambiar la configuración sin modificar el código, facilitando así la adaptabilidad de la aplicación a diferentes entornos (desarrollo, pruebas, producción).

### 3. Backing services:

- Se utilizan servicios externos, como bases de datos y Redis, configurados desde la infraestructura en AWS. Esto permite que los componentes de la aplicación sean independientes y que se puedan gestionar, escalar y actualizar sin afectar al resto del sistema.

### 4. Build, release, run:

- El proceso de construcción, liberación y ejecución de la aplicación está bien definido, permitiendo una clara separación entre estos pasos y facilitando un despliegue continuo.

### 5. Processes:

- La aplicación está diseñada para ser ejecutada en uno o más procesos stateless, lo que significa que no se mantiene estado en la memoria del servidor entre las solicitudes.

### 6. Port binding:

- La aplicación exporta servicios a través de un puerto específico, permitiendo que se pueda ejecutar de forma independiente y que otros servicios puedan interactuar con ella.

### 7. Concurrency:

- La aplicación puede escalar y manejar múltiples instancias de procesos concurrentes, lo que mejora la capacidad de respuesta y la disponibilidad del sistema.

### 8. Disposability:

- La aplicación está diseñada para iniciar y detenerse rápidamente, lo que permite un despliegue ágil y la capacidad de recuperarse de fallos.

### 9. Dev/prod parity:

- Se hace un esfuerzo consciente por mantener la paridad entre los entornos de desarrollo y producción, asegurando que los cambios se prueben en un entorno lo más similar posible al de producción.

### 10. Logs:

- La aplicación genera logs de salida estándar, lo que facilita la supervisión y el análisis de la actividad del sistema.

### 11. Admin processes:

- Los procesos administrativos, como migraciones de base de datos y scripts de mantenimiento, se pueden ejecutar de manera ad-hoc en el entorno de producción.

### 12. Statelessness:

- Se evita el uso de sesión en el servidor, utilizando tokens para la autenticación y almacenamiento de información en el cliente cuando sea necesario.

## 5. Descripción del proceso de Deployment

### Plataforma

Para el despliegue de la aplicación, se utiliza Amazon Web Services (AWS), aprovechando varios de sus servicios clave:

- Elastic Beanstalk: Facilita la implementación y escalado de aplicaciones web. Permite gestionar automáticamente la infraestructura subyacente, lo que permite a los desarrolladores centrarse en escribir código en lugar de preocuparse por la configuración del servidor.
- RDS (Relational Database Service): Utilizado para la gestión de bases de datos relacionales. AWS RDS simplifica la configuración, operación y escalado de bases de datos en la nube, asegurando alta disponibilidad y respaldo automático.
- ElastiCache: Utilizado para mejorar el rendimiento de la aplicación mediante el almacenamiento en caché de datos en memoria. Esto ayuda a reducir la latencia y a mejorar la respuesta de la aplicación, especialmente para datos que se consultan con frecuencia.

La elección de AWS se justifica por su escalabilidad y confiabilidad, permitiendo manejar picos de tráfico y ofreciendo opciones robustas para asegurar la disponibilidad y el rendimiento de la aplicación.

### Pasos para el Despliegue

1. Creación de Instancias:
  - Inicia sesión en tu consola de AWS.
  - Navega a Elastic Beanstalk y crea una nueva aplicación.
  - Selecciona la plataforma deseada (Docker, en este caso) y configura el entorno.
2. Carga del Código:
  - En tu entorno de Elastic Beanstalk, selecciona la opción para cargar el código.
  - Puedes subir tu aplicación directamente como un archivo ZIP o conectarte a un repositorio de código fuente como GitHub.
  - Elastic Beanstalk se encargará de compilar el código y lanzar la aplicación en el entorno configurado.
3. Configuración de Variables de Entorno:
  - En la consola de Elastic Beanstalk, ve a la configuración de tu entorno y agrega las variables de entorno necesarias (por ejemplo, credenciales de base de datos).
4. Configuración de RDS:
  - Ve a RDS y crea una nueva base de datos.
  - Elige el motor de base de datos (MySQL).
  - Configura la instancia, el tamaño, las credenciales de acceso y las opciones de red.
  - Asegúrate de habilitar la conectividad entre tu instancia de Elastic Beanstalk y RDS.
5. Configuración de ElastiCache:
  - Ve a ElastiCache y crea un clúster de Redis.

- Configura los parámetros necesarios, como el tamaño del nodo y las opciones de seguridad.
- Asegúrate de que tu aplicación pueda conectarse al clúster de Redis utilizando el endpoint proporcionado.

## Proceso de Producción

- **Despliegue Manual:**

Actualmente, el proceso de despliegue es manual. Cada vez que se necesita actualizar la aplicación, se sigue el proceso mencionado anteriormente para cargar la nueva versión del código en Elastic Beanstalk.

Esto implica:

- Crear un archivo ZIP con la nueva versión del código.
- Cargarlo a través de la consola de Elastic Beanstalk.
- Configurar las variables de entorno necesarias y realizar las pruebas pertinentes.

- **Migraciones:**

Las migraciones de la base de datos se gestionan a través de scripts que se ejecutan manualmente cuando se despliega una nueva versión, asegurando que la base de datos esté siempre actualizada con la última estructura requerida por la aplicación.

- **Downtime:**

El proceso de despliegue manual puede generar un breve periodo de downtime, ya que la aplicación puede no estar disponible durante el tiempo que tarda en desplegarse y reiniciarse. Se recomienda coordinar los despliegues en horarios de menor tráfico para minimizar el impacto en los usuarios.