

# Trabajo Práctico 1

## Grupo

[75.73/TB034] Arquitectura del Software  
Segundo cuatrimestre de 2025

Alumno	Padrón	Email
CASTRO MARTINEZ, Jose Ignacio	106957	jcastrom@fi.uba.ar
DEALBERA, Pablo Andres	106858	pdealbera@fi.uba.ar
FIGUEROA RODRIGUEZ, Andrea Emperatriz	110450	afigueroa@fi.uba.ar
RICALDI REBATA, Brayan Alexander	103344	bricaldi@fi.uba.ar

# Índice

<b>1. Introducción</b>	<b>2</b>
1.1. Contexto (startup arVault).	2
1.2. Objetivos del TP.	2
1.3. Alcance del análisis.	2
<b>2. QAs identificados</b>	<b>2</b>
2.1. QAs clave y justificación.	2
2.1.1. Disponibilidad	2
2.1.2. Escalabilidad (Elasticidad)	2
2.1.3. Visibilidad	2
2.1.4. Seguridad	2
2.2. Caso base y como cumplen estos QAs claves	2
2.2.1. Disponibilidad	2
2.2.2. Escalabilidad (Elasticidad)	2
2.2.3. Visibilidad	3
2.2.4. Seguridad	3
<b>3. Arquitectura base</b>	<b>3</b>
3.1. Diagrama C&C inicial.	3
3.2. Análisis de decisiones del dev original y su impacto en QA.	4
3.3. Crítica.	4
<b>4. Metodología de pruebas</b>	<b>4</b>
4.1. Herramientas usadas (Artillery, Grafana, etc.).	4
4.2. Escenarios de carga diseñados.	4
4.3. Métricas recolectadas.	4
<b>5. Resultados – Caso base</b>	<b>4</b>
5.1. Screenshots de dashboards.	4
5.2. Interpretación de resultados.	4
<b>6. Propuestas de mejora</b>	<b>4</b>
6.1. Tácticas aplicadas.	4
6.2. Justificación de por qué se eligieron.	4
6.3. Diagramas C&C modificados.	4
<b>7. Resultados – Casos mejorados</b>	<b>4</b>
7.1. Screenshots comparativos.	4
7.2. Análisis de impacto sobre QA.	4
7.3. Trade-offs detectados.	4
<b>8. Conclusiones</b>	<b>4</b>
8.1. Resumen de hallazgos.	4
8.2. Beneficios y limitaciones de las mejoras.	4
8.3. Futuro (ej: pasar a DB externa en TP2).	4
<b>9. Anexos</b>	<b>4</b>
9.1. Configuraciones de escenarios de Artillery.	4
9.2. Configs modificadas de Nginx, Docker Compose, etc.	4

## **1. Introducción**

### **1.1. Contexto (startup arVault).**

### **1.2. Objetivos del TP.**

### **1.3. Alcance del análisis.**

## **2. QAs identificados**

### **2.1. QAs clave y justificación.**

#### **2.1.1. Disponibilidad**

Al ser un servicio de exchange de monedas, asumimos que es un servicio que se utiliza durante todos los días hábiles de la semana en horario cambiario. Por lo tanto, es importante que el servicio se encuentre disponible durante esos horarios para no perder clientes.

También asumimos que los reclamos pueden venir que los clientes no puedan acceder al servicio o que no puedan realizar operaciones.

#### **2.1.2. Escalabilidad (Elasticidad)**

El servicio debe poder escalar para poder atender la demanda de los clientes, que asumimos debe tener picos de demanda importantes en ciertos momentos del día (por ejemplo, en la apertura y cierre del horario cambiario) y momentos de baja o nula demanda, por lo que también sería deseable que el servicio pueda escalar a cero para evitar costos innecesarios.

#### **2.1.3. Visibilidad**

Es importante poder monitorear el estado del sistema para detectar posibles problemas y poder solucionarlos antes de que afecten a los clientes. Además, es importante poder monitorear el rendimiento del sistema para poder escalarlo adecuadamente y evitar problemas de disponibilidad.

#### **2.1.4. Seguridad**

Es fundamental que el sistema sea seguro para evitar posibles ataques que puedan comprometer la integridad del sistema, la privacidad de los datos de los clientes o pérdida/robo de dinero. Incluso pensando que debemos también tener en cuenta marcos regulatorios sobre el manejo de datos personales y financieros. También entendemos del enunciado que es importante la reputación del sistema, y esto podría verse muy dañado en caso de que haya una brecha de seguridad.

### **2.2. Caso base y como cumplen estos QAs claves**

#### **2.2.1. Disponibilidad**

#### **2.2.2. Escalabilidad (Elasticidad)**

Actualmente hay un Nginx que actúa como reverse proxy y potencialmente balanceador de carga, pero en este momento solo tiene configurado una sola instancia de la app de Node.js. De todas formas, notamos varios problemas con esto. En principio, la app es *stateful* porque guarda el estado en memoria y guarda cada tantos segundos el estado de la memoria en distintos archivos json en la carpeta *state/*. Esto hace que no se pueda escalar horizontalmente la app sin perder el estado, ya que cada instancia tendría su propio estado en memoria y no habría forma de sincronizarlos.

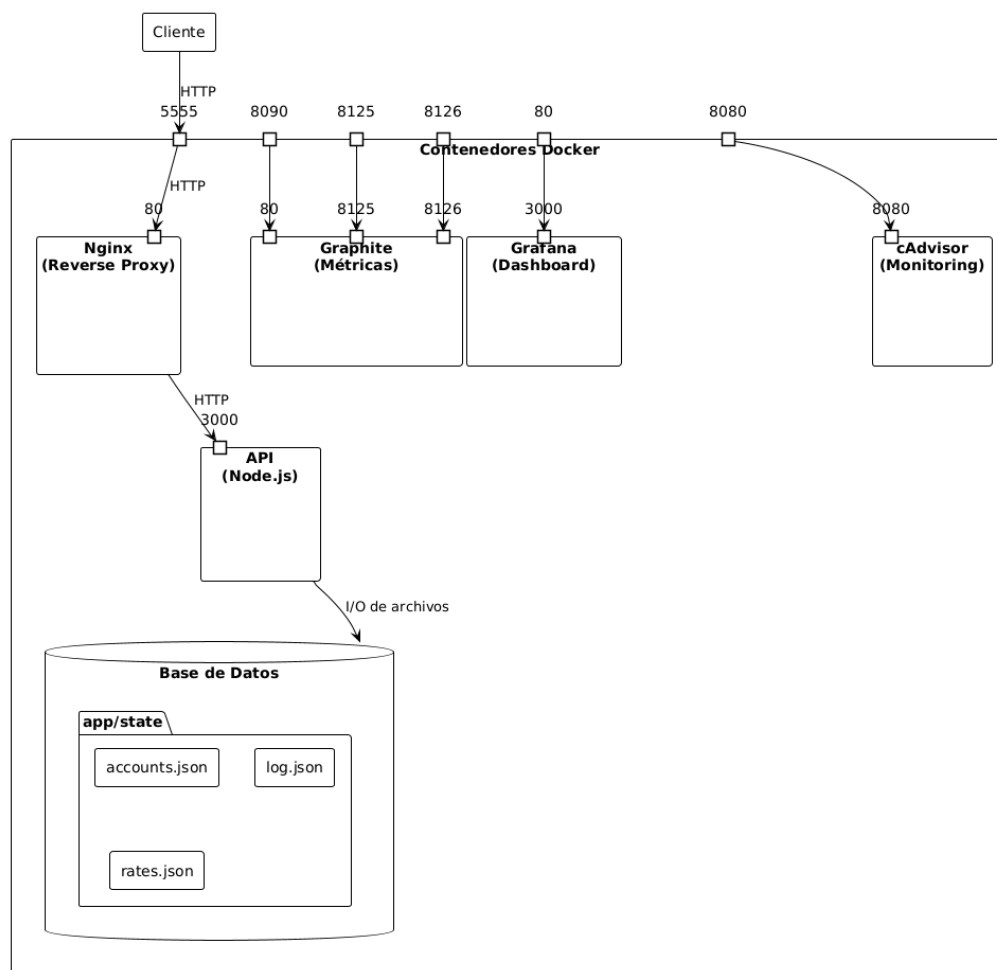
### 2.2.3. Visibilidad

Actualmente hay un contenedor de Graphite y otro de Grafana para monitorear el sistema, y tienen algunas metricas en un dashboard creado por la catedra que permite visualizar algunas metricas como Scenarios launched, Request state, Response time y Resources. Faltarian metricas mas especificas del negocio como por ejemplo, volumen de transacciones por moneda, cantidad de clientes activos, etc.

### 2.2.4. Seguridad

## 3. Arquitectura base

### 3.1. Diagrama C&C inicial.



3.2. Análisis de decisiones del dev original y su impacto en QA.

3.3. Crítica.

## 4. Metodología de pruebas

4.1. Herramientas usadas (Artillery, Grafana, etc.).

4.2. Escenarios de carga diseñados.

4.3. Métricas recolectadas.

## 5. Resultados – Caso base

5.1. Screenshots de dashboards.

5.2. Interpretación de resultados.

## 6. Propuestas de mejora

6.1. Tácticas aplicadas.

6.2. Justificación de por qué se eligieron.

6.3. Diagramas C&C modificados.

## 7. Resultados – Casos mejorados

7.1. Screenshots comparativos.

7.2. Análisis de impacto sobre QA.

7.3. Trade-offs detectados.

## 8. Conclusiones

8.1. Resumen de hallazgos.

8.2. Beneficios y limitaciones de las mejoras.

8.3. Futuro (ej: pasar a DB externa en TP2).

## 9. Anexos

9.1. Configuraciones de escenarios de Artillery.

9.2. Configs modificadas de Nginx, Docker Compose, etc.