Proyecto Unidad 2 y 3: Despliegue y aseguramiento de microservicios con Docker

Administración de Redes y Sistemas Computacionales

Prof. Ricardo Pérez
riperez@utalca.cl

I Semestre, 2025

Introducción

El objetivo de este proyecto es aplicar los conocimientos prácticos de administración de sistemas modernos en un entorno controlado. En la actualidad, muchas organizaciones requieren aplicaciones colaborativas que permitan gestionar tareas y equipos de trabajo de manera eficiente, segura y escalable. Sin embargo, implementar este tipo de sistemas presenta múltiples desafíos, como el diseño de una arquitectura flexible, la integración de microservicios, el aseguramiento de la infraestructura frente a ataques y la necesidad de alta disponibilidad. Por ello, se hace necesario desarrollar una aplicación de gestión de tareas colaborativa, desplegada como microservicios sobre Docker, que permita experimentar y aplicar técnicas modernas de administración, orquestación y aseguramiento en entornos reales y simulados.

El proyecto se divide en **dos unidades evaluativas**:

- **Unidad 2:** Se centra en la contenerización con Docker y la orquestación de servicios con Docker Compose.
- **Unidad 3:** Se enfoca en el aseguramiento de los servicios, el análisis de vulnerabilidades y la implementación de alta disponibilidad.

Modalidad de trabajo y evaluación

- El proyecto se realizará de forma **individual**.

- La evaluación consistirá en una **presentación y defensa** del proyecto en las fechas definidas durante la clase de presentación del proyecto. **No se requiere la entrega de un informe escrito**.
- Durante la presentación, se realizarán preguntas para validar la comprensión de todos los conceptos aplicados. El estudiante debe demostrar dominio completo sobre el trabajo realizado.

Arquitectura de la aplicación

Deberán implementar una arquitectura de microservicios con los siguientes componentes:

1. **Servicio de usuarios (`user-service`):** Gestiona el registro y consulta de usuarios del sistema.
2. **Servicio de tareas (`task-service`):** Maneja la creación, actualización y consulta de tareas asociadas a usuarios.
3. **API Gateway (`nginx-proxy`):** Punto de entrada único que enruta el tráfico hacia los microservicios de forma segura.

---

1. Unidad 2: Construcción y orquestación de microservicios

Objetivo: Contenerizar y orquestar una aplicación funcional de microservicios.

1.1. Servicio de usuarios (`user-service`) (35 puntos)

Este microservicio gestiona toda la información relacionada con los usuarios del sistema.

Requisitos:

- **Funcionalidades principales:** Registro de usuarios, consulta y validación básica de datos.
- **Almacenamiento:** Guardar información de usuarios en una base de datos **SQLite** con esquema apropiado.
- **API REST:** Exponer endpoints para gestión de usuarios:
  - `POST /users` – Registrar un nuevo usuario
  - `GET /users` – Listar todos los usuarios
  - `GET /users/{id}` – Obtener usuario específico
  - `GET /health` – Estado del servicio
- **Validación:** Campos obligatorios, email único, formato de datos apropiado.
- **Contenerización:** Crear un `Dockerfile` optimizado para construir la imagen de este servicio.

1.2. Servicio de tareas (`task-service`) (35 puntos)

Este microservicio maneja toda la lógica relacionada con las tareas del sistema.

Requisitos:

- **Comunicación interna:** Debe comunicarse con el `user-service` para validar asignaciones de tareas.
- **API REST:** Exponer endpoints para gestión de tareas:
  - `POST /tasks` – Crear una nueva tarea
  - `GET /tasks` – Listar todas las tareas
  - `GET /tasks/{id}` – Obtener tarea específica
  - `PUT /tasks/{id}` – Actualizar estado de una tarea
  - `GET /tasks?user_id=X` – Filtrar tareas por usuario
  - `GET /health` – Estado del servicio
- **Estados de tareas:** `pendiente`, `en_progreso`, `completada`.
- **Almacenamiento:** Base de datos **SQLite** con relaciones apropiadas entre usuarios y tareas.
- **Contenerización:** Crear un **`Dockerfile`** para este servicio.

1.3. Orquestación con Docker Compose (30 puntos)

Deberán crear un archivo **`docker-compose.yml`** que defina y conecte todos los servicios.

Requisitos:

- Definir los microservicios creados (`user-service` y `task-service`).
- Establecer una red interna para que se comuniquen de forma segura.
- Configurar el API Gateway (Nginx Proxy Manager) como punto de entrada.
- **Enrutamiento requerido:**
  - `/api/users/*` → Servicio de usuarios
  - `/api/tasks/*` → Servicio de tareas
  - `/admin` → Interfaz del proxy Inverso. Se sugiere utilizar alternativas como Nginx Proxy Manager
- Configurar variables de entorno y volúmenes para persistencia.
- Health checks para todos los servicios.

2. Unidad 3: Seguridad y alta disponibilidad

Objetivo: Asegurar la aplicación, analizar su resiliencia y prepararla para una mayor carga.

2.1. API Gateway seguro con certificados SSL/TLS (35 puntos)

Para resolver este ejercicio es necesario configurar un proxy inverso como **Nginx Proxy Manager** que se desempeñe como API Gateway para centralizar el acceso y añadir una capa de seguridad.

Requisitos:

- **Proxy inverso:** Configurar Nginx para redirigir las peticiones hacia los microservicios correspondientes.
- **Implementación de SSL/TLS:** Asegurar la comunicación mediante HTTPS.
  - Generar e instalar un certificado SSL válido.
  - **Opciones:** Let's Encrypt con Certbot (recomendado) o certificados autofirmados con OpenSSL.
  - Forzar la redirección de todo el tráfico del puerto 80 (HTTP) al 443 (HTTPS).
- **Configuración de seguridad:** TLS 1.2+, configuración robusta de cifrado.
- **Evidencia:** Captura de navegador mostrando conexión segura.
- **Nota práctica:** Para certificados validados externamente, se sugiere el uso de una **instancia en un proveedor Cloud (GCP, AWS, Azure)**.

2.2. Análisis y mitigación de ataques de denegación de servicio (DoS) (35 puntos)

Evaluarán la vulnerabilidad de la aplicación frente a ataques de capa 7.

Requisitos:

- **Herramientas de ataque:** Utilizar `slowhttptest`, o `ab` (Apache Benchmark) para lanzar ataques DoS contra el API Gateway.
- **Análisis de vulnerabilidades:**
  - Ejecutar ataques contra los servicios desplegados
  - Documentar comportamiento durante el ataque (screenshots, logs)
  - Identificar puntos de falla y degradación del rendimiento
- **Implementación de contramedidas (sugeridas):**
  - Rate limiting en Nginx (límites de requests/segundo)

- Configuración de timeouts apropiados
- Límites de conexiones concurrentes
- Monitoring de recursos del sistema

- **Evidencia:** Comparación antes/después de implementar mitigaciones, mostrando la efectividad de las medidas aplicadas.
- **Análisis teórico:** Explicar qué técnicas usaría un SysAdmin para **detectar** que está bajo un ataque DoS en un entorno real.

2.3. Alta disponibilidad con réplicas (30 puntos)

En este ejercicio es necesario mejorar la resiliencia de la aplicación implementando réplicas de los servicios.

Requisitos:

- Modificar el archivo `docker-compose.yml` para desplegar al menos **2 réplicas** de cada microservicio.
- **Pruebas de resiliencia:**
  - Simular falla de una réplica (detener contenedor)
  - Verificar continuidad del servicio
  - Documentar tiempo de recuperación
  - Demostrar balanceo de carga efectivo
- **Evidencia:** Volver a lanzar ataques DoS y demostrar con evidencias que la aplicación es más resiliente y puede gestionar mejor la carga o el fallo de instancias.

Criterios de evaluación

Unidad 2 – Contenerización

- **Funcionalidad (35 %):** Servicios funcionan correctamente, APIs responden adecuadamente.
- **Docker y Compose (40 %):** Dockerfiles optimizados, `docker-compose.yml` bien estructurado.
- **Comunicación entre servicios (20 %):** Microservicios se comunican correctamente.
- **Documentación (5 %):** README claro con instrucciones de despliegue. El repositorio de Github completo con el README detallado si es necesario entregarlo al link de Educandus.

Unidad 3 – Seguridad

- **SSL/TLS (35 %):** Implementación correcta y funcional de certificados.
- **Seguridad DoS (35 %):** Análisis completo y mitigaciones efectivas documentadas.
- **Alta disponibilidad (30 %):** Réplicas funcionando correctamente, pruebas de failover exitosas.

Entregables

1. **Repositorio Git** con código fuente completo y documentación.
2. **Aplicación desplegada** funcionando completamente.
3. **Evidencias de seguridad:** Screenshots, configuraciones, logs de ataques y mitigaciones.
4. **Demostración práctica** durante la defensa oral.

Fechas de defensa

- **Defensa proyecto (Unidad 2):** Última semana de junio 2025.
- **Defensa proyecto (Unidad 3):** Primera semana de julio 2025.

Recursos recomendados

- **Plataformas Cloud:** Google Cloud Platform, AWS, Azure (para certificados SSL válidos)
- **Herramientas:** Docker Desktop, Postman, Kali Linux, SQLite Browser
- **Documentación:** Docker Docs, Nginx Docs, Let's Encrypt