Universidad San Carlos de Guatemala
Facultad de ingeniería.
Ingeniería en ciencias y sistemas

Título del Proyecto:

Delivereats

PONDERACIÓN: 20

Horas Aproximadas: 45

Índice

Contenido

Contenido

1. Resumen del proyecto 3
2. Competencias que desarrollaremos 3
3. Objetivos del Aprendizaje 3
3.1 Objetivo General 3
3.2 Objetivos Específicos 3
4. Enunciado del Proyecto 4
4.1 Descripción del problema a resolver 4
4.2 Alcance del proyecto 4
4.3 Requerimientos técnicos 6
4.4 Entregables 6
5. Desarrollo de Habilidades Blandas 7
5.1 Proyecto en Parejas 7
5.1.1 Trabajo en equipo 7
5.1.2 Responsabilidad y Compromiso 7
5.1.3 Resolución de Problemas 7
6. Cronograma 7
7. Rúbrica de Calificación 8
7.1 Requisitos para optar a la calificación 8
7.2 Detalle de la Calificación 9
7.3 Valores 11
1. Originalidad del Trabajo: 11
2. Prohibición de Copias y Plagio 11
3. Uso Responsable de Recursos Externos 11
4. Revisión y Validación del Trabajo 11
5. Entrega y Fechas Límite 11

1. Resumen del proyecto

En esta segunda fase del proyecto "Delivereats", el enfoque evoluciona desde la construcción de una arquitectura base hacia la implementación de un sistema resiliente, escalable y con procesos de despliegue profesionales. Se integrarán patrones de arquitectura avanzados para desacoplar servicios críticos y gestionar el ciclo de vida de la aplicación en producción.

El sistema incorporará orquestación mediante Kubernetes, comunicación asíncrona mediante colas de mensajes y estrategias de despliegue seguro (Rollout/Rollback), elevando el nivel de ingeniería hacia estándares de la industria.

2. Competencias que desarrollaremos

- **Orquestación de Contenedores Avanzada:** Capacidad para gestionar aplicaciones en Kubernetes utilizando estrategias de actualización sin tiempo de inactividad (Zero Downtime).
- **Arquitectura Orientada a Eventos:** Implementación de desacoplamiento de servicios mediante colas de mensajes (RabbitMQ/Kafka) para garantizar la tolerancia a fallos.
- **DevOps y CI/CD:** Diseño de pipelines para automatizar pruebas y despliegues.
- **Integración de Servicios Financieros:** Manejo de conversiones monetarias y transacciones.
- **Aseguramiento de la Calidad:** Implementación de pruebas unitarias.

3. Objetivos del Aprendizaje

3.1 Objetivo General

Evolucionar la plataforma desarrollada en la Fase 1, transformándola en un sistema distribuido de alta disponibilidad mediante Kubernetes, comunicación asíncrona y la implementación de estrategias de despliegue que permitan actualizaciones y reversiones (Rollback) controladas.

3.2 Objetivos Específicos

- Implementar un clúster de Kubernetes que gestione la disponibilidad de los microservicios.
- **Diseñar y demostrar una estrategia de despliegue (Rollout) y recuperación (Rollback)** ante fallos en una nueva versión.
- Sustituir la comunicación síncrona crítica entre órdenes y restaurantes por un sistema de colas (RabbitMQ/Kafka).
- Desarrollar servicios financieros (Pagos, FX con caché) y sistemas de retroalimentación (Calificaciones).
- Configurar un pipeline de CI/CD para automatizar el testing y el despliegue.

4. Enunciado del Proyecto

4.1 Descripción del problema a resolver

Tras el MVP de la Fase 1, se detectó que las actualizaciones del sistema causaban caídas del servicio (Downtime) y que, si una nueva versión tenía errores, no existía una forma rápida de volver a la versión anterior. Además, la dependencia directa (gRPC) entre servicios hacía que el fallo de uno detuviera toda la operación.

Para resolver esto, se requiere una arquitectura que soporte **actualizaciones progresivas** y comunicación asíncrona, permitiendo que el negocio siga operando incluso durante despliegues o picos de carga..

4.2 Alcance del proyecto

Payment-Service & Cupones

- Procesamiento de pagos simulados (Tarjeta Crédito/Débito).
- Cartera digital recargable
- **Cupones:** Sistema de descuentos validables.
- Comunicación con Order-Service para confirmar el estado "PAGADO".

FX-Service (Conversion Service)

El sistema debe manejar precios internamente en quetzales y poder mostrar al cliente precios en monedas como USD/MXN/JPY u otras equivalentes según disponibilidad.

- Cambiar USD por otra base (ej. EUR): https://open.er-api.com/v6/latest/EUR

Reglas mínimas del fx-service:

- **API Externa:** Debe consumir una API real de tipo de cambio (ej. ExchangeRate-API, OpenExchangeRates) para obtener el valor del día.
- **Caché/Fallback:** Para evitar consumo excesivo de la API externa y mejorar latencia, se debe implementar una caché (Redis) que almacene la tasa de cambio por un tiempo definido (ej. 12 horas). Si la API falla, se usa el valor en caché.

Sistema de Calificaciones

Permite a los usuarios calificar tres entidades tras finalizar una orden:

- Al **Repartidor** (1-5 estrellas + comentario).
- Al **Restaurante** (1-5 estrellas + comentario).
- Al **Producto** específico (Recomendado o no recomendado)

Cola de Pedidos (RabbitMQ / Kafka)

La creación de pedidos hacia el restaurante ya no será por gRPC directo.

- Order-Service publicará el pedido en una cola
- Restaurant-Service consumirá el mensaje a su propio ritmo. Esto evita que el sistema de órdenes falle si el sistema de restaurantes está saturado.

Delivery-Service (ACTUALIZACIÓN):

- **Evidencia de Entrega:** Al marcar un pedido como "ENTREGADO", el repartidor debe subir obligatoriamente una **fotografía** que sirva como prueba de entrega.

Kubernetes: despliegue y operación

El repositorio debe contener un directorio claramente identificable (por ejemplo k8s/ o deploy/k8s/) con los manifiestos necesarios para describir el despliegue. Como mínimo:

- Un Deployment por cada componente que se va a desplegar en el clúster (frontend, gateway/BFF, servicio de autenticación, servicio de negocio seleccionado, etc.).
- Un Service por cada Deployment, de forma que cada microservicio tenga un nombre DNS estable dentro del clúster.
- Uno o varios recursos Ingress para exponer hacia Internet el frontend y el punto de entrada de la API (gateway/BFF o servicio público), con rutas claras y, si se aplica, prefijos de versión.
- Uno o varios ConfigMaps para agrupar configuración no sensible:
  - direcciones de otros servicios dentro del clúster
  - URLs de APIs externas
  - timeouts por defecto
  - flags de características, etc.
- Uno o varios Secrets para agrupar información sensible:
  - secretos usados para firmar/verificar JWT,
  - credenciales de base de datos,
  - credenciales del registry de contenedores,
  - cualquier otro valor que no deba aparecer en texto plano.

Los manifiestos deben ser completos y consistentes: un Deployment que referencia un ConfigMap o Secret debe indicar claramente los env o envFrom correspondientes; los Servicios deben apuntar a los labels correctos; los Ingress deben apuntar a Servicios existentes, etc

CI/CD: de la imagen al despliegue

En esta fase el pipeline deja de ser únicamente de integración continua y se extiende a un ciclo de integración y despliegue.

1. **Build**
   - Construir las imágenes Docker de los servicios relevantes (microservicios y, si aplica, frontend).
   - Seguir una convención de tags que permita identificar la versión y, cuando aplique, la rama de origen.

2. **Test**
   - Ejecutar pruebas automáticas (unitarias u otras que el equipo tenga definidas).
   - Detener el pipeline si alguna prueba falla.

3. **Publicación**
   - Publicar las imágenes construidas en un registry.
   - Usar credenciales inyectadas mediante variables de entorno o secretos del sistema de CI/CD (no deben aparecer en el repositorio).

4. **Despliegue a Kubernetes**
   - Ejecutar los comandos necesarios para actualizar el clúster (por ejemplo, kubectl apply -f k8s/).

- Dejar evidencia en los logs del pipeline de qué se desplegó y con qué versión.

Publicación en registry

La publicación debe cubrir, al menos, las imágenes que se usan en el clúster. Se espera que:

- Los nombres de imagen sigan una convención coherente (por ejemplo, <registry>/<grupo>/servicio:<version>).
- El pipeline utilice un usuario/clave o token configurado como secreto del sistema de CI/CD.

Despliegue automatizado a Kubernetes

La etapa de despliegue debe:

- Ser reproducible (un commit con el mismo pipeline debe producir el mismo efecto, salvo cambios externos).
- Aplicar los manifests del repositorio, no manifests distintos almacenados fuera del control de versión.
- Dejar en los logs:
  - el clúster/entorno al que se desplegó,
  - el resultado de la aplicación de manifests (éxito/errores).

4.3 Requerimientos técnicos

Herramientas	Uso requerido
Kubernetes (K8s)	Orquestación de todos los microservicios
Ingress Controller	Punto de entrada único (Nginx o similar) para gestionar el tráfico hacia el clúster.
RabbitMQ / Kafka	Gestión de colas de mensajería asíncrona.
Redis	Caché para el servicio de conversión de moneda.
CI/CD	GitHub Actions, GitLab CI o Jenkins para pipelines de despliegue y test.
Unit Testing	JUnit, Jest, PyTest (según lenguaje). Cobertura mínima del 70% en servicios core
Docker, Docker Compose	Contenedores

4.4 Entregables

- **Documentación completa**
- **Código fuente de la aplicación:** Repositorio actualizado con las nuevas ramas de desarrollo
- **Archivos de configuración K8s:** Manifiestos (.yaml) para Deployments, Services, Ingress, ConfigMaps y Secrets
- **Pipeline CI/CD:** Archivo de configuración del pipeline
- **Reporte de Cobertura:** Screenshots o reporte generado por la herramienta de testing
- **Documentación actualizada:** Actualización de toda la documentación del proyecto
- **Nombre del repositorio:** *SA_PROYECTO_Carnet*.
- **Medio de entrega:** A través de la plataforma UEDI

5. Desarrollo de Habilidades Blandas

5.1 Proyecto en Parejas

El trabajo en equipo es clave para el éxito de cualquier proyecto, por lo que la comunicación con la pareja será vital para el desarrollo y éxito al final del proyecto
Además, se fomenta la toma de decisiones conjunta y la resolución de problemas en grupo, desarrollando habilidades como la negociación. Los grupos recibirán retroalimentación para mejorar sus soluciones y adaptarse a los desafíos que surjan.

5.1.1 Trabajo en equipo

Cada grupo debe trabajar en colaboración, asignando roles específicos a cada miembro (líder de proyecto, desarrollador, diseñador, etc.). Los estudiantes deben coordinarse mediante herramientas de gestión como Trello, jira, etc..

5.1.2 Responsabilidad y Compromiso

Los grupos deben realizar presentaciones periódicas ante sus compañeros y el profesor, compartiendo los avances y problemas encontrados, y recibiendo retroalimentación para mejorar sus soluciones.

5.1.3 Resolución de Problemas

A lo largo del proyecto, los estudiantes pueden enfrentar desafíos o desacuerdos. Deberán aplicar técnicas de resolución de conflictos para mantener la armonía y la productividad en el equipo.

6. Cronograma

Tipo	Fecha Inicio	Fecha Fin
Asignación de Proyecto	19/02/2026	19/02/2026
Elaboración	19/02/2026	19/03/2026
Calificación	20/03/2026	21/02/2026

7. Rúbrica de Calificación

7.1 Requisitos para optar a la calificación

Antes de ser evaluado, el proyecto deberá cumplir con los siguientes requisitos mínimos. Si alguno de estos no se cumple, el proyecto no podrá ser calificado y se considerará como no entregado o incompleto:

Tema	Descripción	Cumple (Sí/No)
Gestión y entregas del proyecto	Se deben haber entregado y calificado las practicas 4, 5 y 6 del laboratorio para tener derecho a calificación de la fase 1	

7.2 Detalle de la Calificación

Descripción	Valor	Punteo
Nuevas funcionalidades	**30**	**-**
Sistema de metodos de pago y cupones	2.5	
Conversiones de Moneda (API + REDIS/CACHE)	10	
Sistema de calificaciones (Repartidor/Restaurante/Producto)	5	
Evidencia de entrega (Fotrografia)	2.5	
Implementación de Colas de pedidos	10	
Infraestructura y Orquestación (Kubernetes)	**30**	
Despliegue correcto en K8s (Pods/Services/Namespace)	10	
Uso de ingress	5	
Configuración (ConfigMaps/Secrets)	5	
Persistencia (volúmenes para DB o archivos)	5	
Estrategia de Rollout y Rollback	5	
DevOps y Calidad	**20**	
Pipeline CI/CD (Build/Test/Deploy)	10	
Pruebas Unitarias (Ejecución y Cobertura)	10	
Frontend (Actualización)	**10**	
Integración de nuevos flujo (Pagos, Foto, etc.)	10	
Preguntas	**10**	
Pregunta 1	2	
Pregunta 2	2	
Pregunta 3	2	
Pregunta 4	2	
Pregunta 5	2	
Penalizaciones		
No hay despliegue en Kubernetes	-100%	
No se realiza el despliegue por medio de CI/CD	-100%	
No se encuentra utilizando un servicio en la nube	-100%	**-**
No se evidencia el uso de ingress	-50%	
No se implementó el sistema de colas para los pedidos	-25%	

No hay evidencia de Pull Request	-20%	
Documentación desactualizada	-20%	
TOTAL	**100**	

7.3 Valores

En el desarrollo del proyecto se espera que los estudiantes demuestren un alto nivel de honestidad académica, responsabilidad ética y compromiso profesional. El cumplimiento de los principios que se detallan a continuación será obligatorio y cualquier incumplimiento será sancionado conforme a las normativas vigentes de la Escuela de Ciencias y Sistemas.

1. **Originalidad del Trabajo:**
   Cada estudiante o equipo debe desarrollar su propio código y/o documentación, aplicando los conocimientos adquiridos en el curso.

2. **Prohibición de Copias y Plagio**
   Queda estrictamente prohibido copiar, replicar o adaptar total o parcialmente el código, documentación, lógica o cualquier componente del proyecto desde otras fuentes sin el debido análisis, modificación y referencia.

   - La detección de plagio (entre compañeros, de internet o de ciclos anteriores) será penalizada con calificación de 0 puntos.
   - Los responsables serán **reportados formalmente a la Escuela de Ciencias y Sistemas**.
   - Esto incluye el uso de ediciones superficiales de código ajeno, sin comprensión real ni justificación lógica.

3. **Uso Responsable de Recursos Externos**
   Está permitido el uso de bibliotecas de consulta, ejemplos o fragmentos educativos siempre y cuando:

   - Se referencien adecuadamente en el código o en los anexos.
   - Se comprenda completamente su funcionamiento.
   - No se utilicen como sustituto de la lógica propia del proyecto. Cualquier duda deberá ser consultada con el catedrático o auxiliar antes de su uso.

4. **Revisión y Validación del Trabajo**
   El equipo docente podrá utilizar herramientas automáticas y revisiones manuales para comparar entregas y detectar similitudes no justificadas.

   - En caso de sospecha, el estudiante deberá defender su solución, explicar sus decisiones y justificar el funcionamiento de su código.
   - Si no logra demostrar la autoría o comprensión del trabajo, se asignará **una calificación de 0 puntos**, sin opción a apelación académica.

5. **Entrega y Fechas Límite**
   - No se permitirá realizar modificaciones al código ni a la documentación después de la fecha de entrega final establecida en el cronograma.
   - Las entregas fuera del plazo establecido pueden considerarse no válidas y se aplicarán las penalizaciones indicadas, a menos que se haya aprobado una prórroga justificada con anterioridad.