Escuela Superior de Cómputo
Evaluación a Título de Suficiencia Especial

Sistemas Distribuidos

Prototipo de sistema de comercio electrónico utilizando microservicios sobre Kubernetes en la nube

Resumen

Se desarrollará un prototipo de sistema de comercio electrónico desplegado en la nube, consistente en un front-end HTML-Javascript y un back-end implementado como microservicios Java, accediendo a una instancia de MySQL administrada (PaaS). Cada microservicio ejecutará en un pod de Kubernetes.

El servidor web será Nginx y ejecutará en un pod. Los archivos del front-end estarán en un almacenamiento compartido. El servidor web accederá a los archivos del front-end utilizando Persistent Volume (PV) y Persistent Volume Claim (PVC).

El API Gateway será Nginx, el cual recibirá peticiones HTTP y ejecutará en un pod. El API Gateway se expondrá al cliente (navegador) mediante un servicio de Kubernetes de tipo LoadBalancer.

El API Gateway enrutará las peticiones HTTP a los pods que ejecutarán los microservicios (incluyendo el servidor web), por tanto, los pods se expondrán al API Gateway utilizando un servicio de Kubernetes de tipo ClusterIP.

No se requiere escalado automático de pods, por tanto, las replicas de los pods se definirán directamente en los manifest files de tipo Deployment.

Requerimientos no funcionales

1. El back-end consistirá de cuatro microservicios Java ejecutando en Kubernetes administrado (PaaS):

    1. **Gestión de usuarios**. Microservicio para gestionar los datos de los usuarios.
    2. **Gestión de artículos**. Microservicio para gestionar los datos de los artículos.
    3. **Gestión de compras**. Microservicio para gestionar la compra de artículos.
    4. **Servidor web**. Microservicio que ejecutará Nginx en un pod.

2. Se deberá utilizar MySQL administrado en la nube (PaaS).

3. Se deberá utilizar Kubernetes administrado en la nube (PaaS).

4. Los archivos del front-end (.html, .js, .jpeg, .css, etc.) se deberán colocar en un almacenamiento compartido (Azure Files, Amazon EFS o Google Filestore). Los archivos del

front-end no deberán colocarse en el file system del contenedor que ejecuta el servidor web (Nginx).

5. El front-end se desarrollará en HTML-Javascript. Se podrá utilizar cualquier framework (p.e. React, Angular, Bootstrap, etc.).

6. El back-end consistirá en servicios web Java ejecutando en pods de Kubernetes. Se podrá utilizar cualquier framework para desarrollar los microservicios (p.e. Spring Boot, Azure Functions, JDK, etc.).

6.1 Las funciones del back-end invocadas con POST y PUT recibirán JSON en el “body”.

6.2 Todas las funciones del back-end (GET, POST, PUT y DELETE) regresarán JSON.

7. Cada microservicio tendrá su propia base de datos y no deberá acceder a la base de datos de otro microservicio; suponiendo que “12345678” es el número de boleta del alumno o alumna:

7.1 La base de datos para la Gestión de usuarios se llamará “bdgu_12345678” e incluirá la siguiente tabla:

Tabla: usuarios		
Columna	Tipo	Nulo
id_usuario	int	no
email	varchar(100)	no
password	varchar(64)	no
token	varchar(40)	si
nombre	varchar(100)	no
apellidos	varchar(100)	no

El id_usuario será llave primaria auto-incrementada.

La columna “password” almacena el hash (sha-256) de la contraseña en formato hexadecimal. Debido a que sha-256 produce un número de 32 bytes, el password se almacena como 64 caracteres hexadecimales.

Se deberá crear un índice único para el campo “email”.

7.2 La base de datos para la Gestión de artículos se llamará “bdga_12345678” e incluirá las siguientes tablas:

Tabla: stock		
Columna	Tipo	Nulo
id_articulo	int	no
nombre	varchar(256)	no
descripcion	varchar(256)	no
precio	decimal(10,2)	no

El id_articulo será llave primaria auto-incrementada.

Tabla: fotos_articulos		
Columna	Tipo	Nulo
id_foto	int	no
foto	longblob	no
id_articulo	int	no

7.3 La base de datos para la Gestión de compras se llamará “bdgc_12345678” e incluirá las siguientes tablas:

Tabla: carrito_compra		
Columna	Tipo	Nulo
id_usuario	int	no
id_articulo	int	no
cantidad	int	no

Se deberá crear un índice único para los campos (id_usuario,id_articulo).

Tabla: stock		
Columna	Tipo	Nulo
id_articulo	int	no
cantidad	int	no

8. Para cada microservicio se deberá escribir: 1) un Dockerfile para crear la imagen, 2) un manifest file de tipo Deployment para el despliegue de los pods y 2) un manifest file de tipo ClusterIP para el despliegue del servicio que expone el microservicio al API Gateway.

9. Para que el servidor web (Nginx) tenga acceso a los archivos correspondientes a la aplicación web, es necesario crear un servicio Persistent Volume (PV) y un servicio Persistent Volumen Claim (PVC). El PVC deberá definir storageClassName: "" (lo cual indica que el almacenamiento compartido no lo creará automáticamente Kubernetes), por tanto, el almacenamiento compartido deberá ser creado por el alumno o la alumna y deberán cargar los archivos del front-end (html, js, css, jpeg, png, etc.) en el almacenamiento compartido.

10. El API Gateway será Nginx, el cual deberá ejecutar en un pod en Kubernetes y deberá exponer un servicio de Kubernetes de tipo Loadbalancer.

11. El sistema mostrará las siguientes pantallas:

11.1 **Login**. Pantalla que permite ingresar el email del usuario y la contraseña. Así mismo, en esta pantalla se incluirá una opción para registrar un nuevo usuario.

11.2 **Alta de usuario**. Pantalla para registrar un nuevo usuario.

11.3 **Menú principal**. Pantalla que muestra un menú con las siguientes opciones: “Captura de artículo” y “Compra de artículos”.

11.4 **Captura de artículo**. Pantalla para dar de alta los datos de un artículo.

11.5 **Compra de artículos**. Pantalla para buscar artículos y para comprar artículos.

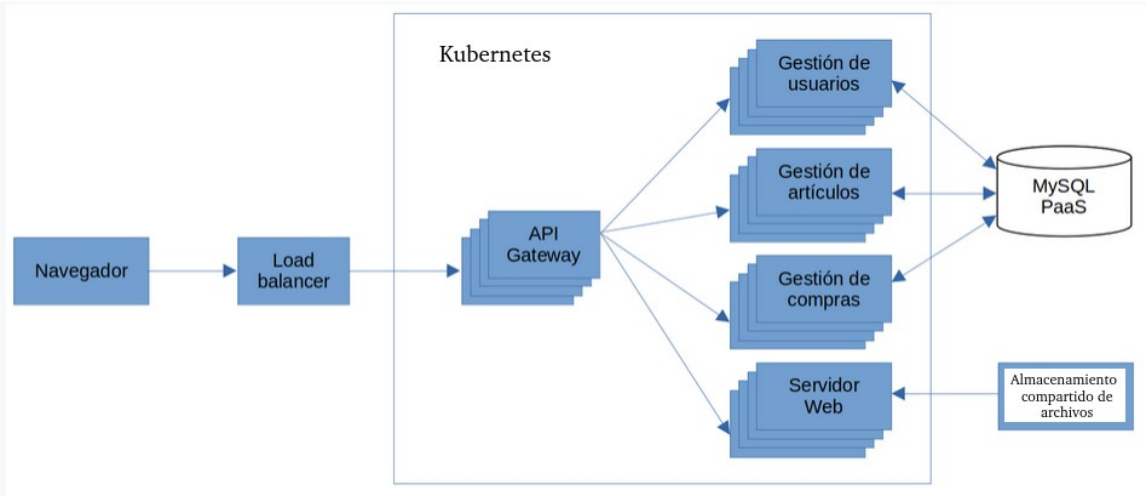
11.6 **Carrito de compra**. Pantalla que muestra los artículos en el carrito de compra.

12. Solo se podrá utilizar Microsoft Azure, AWS o Google Cloud. La siguiente tabla muestra cómo se llaman las diferentes tecnologías a implementar, de acuerdo a la plataforma a utilizar:

	Azure	AWS	Google Cloud
Kubernetes administrado	AKS (Azure Kubernetes Service)	EKS (Elastic Kubernetes Service)	GKE (Google Kubernetes Engine)
Servicio interno entre pods	Serrvicio ClusterIP	Serrvicio ClusterIP	Serrvicio ClusterIP
Exposición externa del API Gateway	Servicio LoadBalancer	Servicio LoadBalancer	Servicio LoadBalancer
Balanceador creado automáticamente	Azure Load Balancer	AWS ELB / NLB	Google Cloud Load Balancer
Almacenamiento compartido de archivos	Azure Files	Amazon EFS	Filestore
Persistent Volume	PV (Azure Files CSI)	PV (EFS CSI)	PV (Filestore CSI)
Persistent Volume Claim	PVC	PVC	PVC
MySQL administrado	Azure Database for MySQL	Amazon RDS for MySQL	Cloud SQL for MySQL
Registro de imágenes de contenedor	Azure Container Registry (ACR)	Amazon ECR	Artifact Registry
Escalado de los microservicios	Replicas en manifest file	Replicas en manifest file	Replicas en manifest file

Arquitectura del sistema

La arquitectura del sistema es la siguiente (notar que podrán existir varias réplicas de los pods que ejecutan el Api Gateway y los microservicios):



Requerimientos funcionales del back-end

1. El microservicio "Gestión de usuarios" deberá incluir las siguientes funciones:

- **usuarios/alta_usuario**. Da de alta un usuario. Recibe como parámetros los datos del usuario: email, hash del password (sha-256), nombre y apellidos. Si no hay error regresa el código HTTP 200 y {“mensaje”:”OK”} de otra manera regresa el código HTTP 400 y {“mensaje”:”<mensaje de error>”}. La petición HTTP deberá ser POST.
- **usuarios/login**. Autentica un usuario. Recibe como parámetros el email y el hash de la contraseña; si el email y el hash de la contraseña se encuentran en la tabla de "usuarios", genera un token aleatorio de 40 caracteres, actualiza el token en la tabla "usuarios" y regresa el código HTTP 200 y {“id_usuario”: <id_usuario>, ”token”: ”<token>”}. Si el email y el hash de la contraseña no se encuentran en la tabla "usuarios" regresa el código HTTP 400 y {“mensaje”: "Acceso denegado"}. La petición HTTP deberá ser POST.
- **usuarios/verifica_acceso**. Verifica que el usuario se haya autenticado (logueado). La función recibirá como parámetros el id_usuario y el token; regresará el código HTTP 200 y {“acceso”:<true o false>}, indicando si el id_usuario y el token se encontraron en la tabla de "usuarios". Si hay error regresa el código HTTP 400 y {“mensaje”: "<mensaje de error"}. La petición HTTP deberá ser GET.

2. El microservicio "Gestión de artículos" deberá incluir las siguientes funciones:

- **articulos/alta_articulo.** Da de alta artículos en la tabla "stock". La función recibirá como parámetros el nombre del artículo, la descripción del artículo, el precio, la cantidad de artículos en existencia, la fotografía del artículo, el id_usuario y el token del usuario. Para ejecutar la función, se deberá verificar el acceso utilizando la función "verifica_acceso" del microservicio "Gestión de usuarios". Debido a que la cantidad del artículo no existe en la base de datos del microservicio "Gestión de artículos", la función "alta_articulo" deberá invocar la función "alta_articulo" del microservicio "Gestión de compras". Si no hay error regresa el código HTTP 200 y {“mensaje”:“OK”} de otra manera regresa el código HTTP 400 y {“mensaje”:”<mensaje de error>”}. El INSERT a la tabla “stock” y el INSERT a la tabla “fotos_artículos” deberá hacerse **dentro de una transacción**. La petición HTTP deberá ser POST.
- **articulos/consulta_articulos.** Busca una palabra (p.e. mayonesa) en los campos “nombre” y “descripcion” en la tabla “stock”. Si no hay error, regresa el código HTTP 200 y un arreglo JSON con los datos de los artículos (id_articulo, fotografía, nombre, descripción y precio). Si hay error regresa el código HTTP 400 y {“mensaje”:”<mensaje de error>”}. La búsqueda se deberá realizar utilizando una instrucción SELECT con LIKE. La función recibirá como parámetros la palabra clave, el id_usuario y el token del usuario. Para ejecutar la función, se deberá verificar el acceso utilizando la función "verifica_acceso" del microservicio "Gestión de usuarios". La petición HTTP deberá ser GET.

3. El microservicio "Gestión de compras" deberá incluir las siguientes funciones:

- **compras/alta_articulo.** Da de alta el artículo en la tabla "stock". La función recibirá como parámetros el id_artículo, la cantidad en existencia, el id_usuario y el token del usuario. Si no hay error regresa {“mensaje”:”OK”} de otra manera regresa {“mensaje”:”<mensaje de error>”}. Para ejecutar la función, se deberá verificar el acceso utilizando la función "verifica_acceso" del microservicio "Gestión de usuarios". La petición HTTP deberá ser POST.
- **compras/compra_articulo.** Realiza la compra de un artículo. La función recibirá como parámetros el id_articulo, la cantidad a comprar, id_usuario y el token del usuario. Si no hay error, regresa el código HTTP 200 y {“mensaje”:”OK”}, de lo contrario, regresa el código HTTP 400 y {“mensaje”:”<mensaje de error>”}. Para ejecutar la función, se deberá verificar el acceso utilizando la función "verifica_acceso" del microservicio "Gestión de usuarios". La petición HTTP deberá ser PUT.
    - Si la cantidad de artículos a comprar es menor o igual a la cantidad de artículos en la tabla "stock" y el artículo no existe en el carrito de compra del usuario, se deberá insertar en la tabla "carrito_compra" el id_usuario, id_articulo y la cantidad, así mismo, se restará la cantidad a comprar de la cantidad en la tabla de "stock". El INSERT a la tabla "carrito_compra" y el UPDATE a la tabla "stock" se deberán realizar **dentro de una transacción**.
    - Si la cantidad de artículos a comprar es menor o igual a la cantidad de artículos en la tabla "stock" y el artículo ya existe en el carrito de compra del usuario, se deberá agregar la cantidad de artículos. El UPDATE a la tabla "carrito_compra" y el UPDATE a la tabla "stock" se deberán realizar **dentro de una transacción**.
    - Si la cantidad de artículos a comprar es mayor a la cantidad de artículos en la tabla "stock", se deberá regresar un código HTTP 400 indicando que hubo error y {“mensaje”:”No hay suficientes artículos en el stock"}.
- **compras/elimina_articulo_carrito_compra.** Elimina un artículo de la tabla "carrito_compra". La función recibirá como parámetros el id_usuario, id_articulo del artículo a eliminar del carrito de compra del usuario y el token del usuario. Si no hay error regresa el código HTTP 200 y {“mensaje”:”OK”} de otra manera regresa el código HTTP 400 y {“mensaje”:”<mensaje de error>”}. Para ejecutar la función, se deberá verificar el acceso utilizando la función "verifica_acceso" del microservicio "Gestión de usuarios". La petición HTTP deberá ser DELETE.
    - La función deberá eliminar el artículo del carrito de compra del usuario, agregando la cantidad de los artículos en la tabla "stock" y borrando el registro correspondiente de la tabla "carrito_compra".
    - La actualización (UPDATE) de la tabla "stock" y el borrado (DELETE) del artículo de la tabla "carrito_compra", deberán realizarse **dentro de una transacción**.
- **compras/elimina_carrito_compra.** Borra todos los registros del carrito_compra del usuario. La función recibirá como parámetros el id_usuario y el token del usuario. Si no hay error regresa el código HTTP 200 y {“mensaje”:”OK”} de otra manera regresa el

código HTTP 400 y {“mensaje”:”<mensaje de error>”}. Para ejecutar la función, se deberá verificar el acceso utilizando la función "verifica_acceso" del microservicio "Gestión de usuarios". La petición HTTP deberá ser DELETE,.

- La función deberá eliminar cada artículo del carrito de compra del usuario, agregando la cantidad de los artículos en la tabla "stock" y borrando los registros de la tabla "carrito_compra" correspondientes al usuario.
- La actualización (UPDATE) de la tabla "stock" y el borrado (DELETE) de la tabla "carrito_compra" deberán realizarse **dentro de una transacción**.

Requerimientos funcionales del front-end

1. La pantalla “Login” es la pantalla inicial. Esta pantalla permite capturar el email y la contraseña del usuario, entonces invoca la función “login” del microservicio “Gestión de usuarios” (ver requerimiento funcional 1 del back-end). No deberá enviarse al back-end la contraseña, en su lugar, se enviará el hash de la contraseña.

Si el email y la contraseña son correctos, se guarda el id_usuario y el token y se despliega la pantalla “Menú principal”. Si el email o la contraseña no son correctos, se despliega el mensaje “Acceso denegado”.

2. La pantalla “Login” incluye una opción para registrar un nuevo usuario. Cuando se selecciona esta opción, se despliega la pantalla “Alta de usuario”.

3. La pantalla “Alta de usuario” permite capturar los datos del usuario (ver requerimiento funcional 1 del back-end).

3. Al seleccionar la opción "Captura de artículo" se deberá desplegar la pantalla "Captura de artículo", la cual permitirá capturar el nombre del artículo, la descripción del artículo, el precio, la cantidad de artículos en existencia y la fotografía del artículo (ver el requerimiento funcional 2 del back-end).

4. Al seleccionar la opción "Compra de artículos" se deberá desplegar la pantalla "Compra de artículos". En esta pantalla el usuario podrá ingresar una palabra que se buscará en los campos "nombre" y "descripción" de la tabla "stock" (ver requerimiento funcional 2 del back-end).

5. Para cada artículo resultado de la búsqueda, se deberá desplegar en la pantalal “Compra de artículos” una pequeña foto del artículo, el nombre, la descripción, el precio, un botón de "Compra" y un campo de "Cantidad" con un valor default igual a 1.

6. Cuando el usuario presione el botón de "Compra", se realizará la compra (ver requerimiento funcional 3 del back-end).

7. La pantalla de "Compra de artículos" deberá disponer de un botón "Carrito de compra" el cual deberá desplegar una pantalla "Artículos en el carrito" con los artículos en la tabla "carrito_compra", incluyendo una pequeña imagen del artículo, nombre del artículo, la cantidad a comprar, precio y costo (cantidad x precio). Así mismo, en la ventana "Artículos en el carrito" se deberá desplegar el total de la compra.

8. Para cada artículo en la pantalla "Artículos en el carrito" se deberá incluir un botón "Eliminar artículo" el cual permitirá eliminar el artículo del carrito de compra (ver requerimiento funcional 3 del back-end).

9. La pantalla "Artículos en el carrito" deberá tener un botón "Eliminar carrito" el cual permitirá borrar el carrito de compra (ver requerimiento funcional 3 del back-end).

10. La pantalla "Artículos en el carrito" deberá tener un botón "Seguir comprando" el cual deberá permitir regresar a la pantalla "Compra de artículos".

Evaluación del proyecto

° Antes de la presentación de su proyecto, los alumnos y alumnas deberán enviar los siguientes archivos al correo electrónico capineda@ipn.mx: 1) código fuente del front-end, 2) código fuente del back-end, 3) archivos de configuración de Nginx como servidor web, 4) archivos de configuración de Nginx como API Gateway, 5) scripts de las bases de datos, 6) docker files y 7) manifest files (cuatro Deployment, cuatro ClusterIP, un LoadBalancer, un PV y un PVC).

° Los alumnos y alumnas deberán presentar su proyecto al profesor evaluador en el laboratorio el día 10 de febrero de 2026 en el horario que corresponda, matutino o vespertino.

° Para la evaluación del proyecto, se utilizará una rúbrica la cual permitirá verificar que se hayan cumplido los requerimientos funcionales y no funcionales descritos en las especificaciones del proyecto.

° Para presentar su proyecto, los alumnos y las alumnas podrán utilizar su propio equipo de cómputo o una computadora del laboratorio.