# Propuesta de Arquitectura y Documentación Técnica

## Sistema de Reservas en la Nube

### 1. Introducción

Este documento describe la arquitectura de software propuesta para el sistema de reservas en la nube, diseñada para cumplir con los requisitos de escalabilidad, seguridad, eficiencia y mantenibilidad. La arquitectura se basa en un enfoque de microservicios para garantizar la flexibilidad y la capacidad de adaptación a futuro.

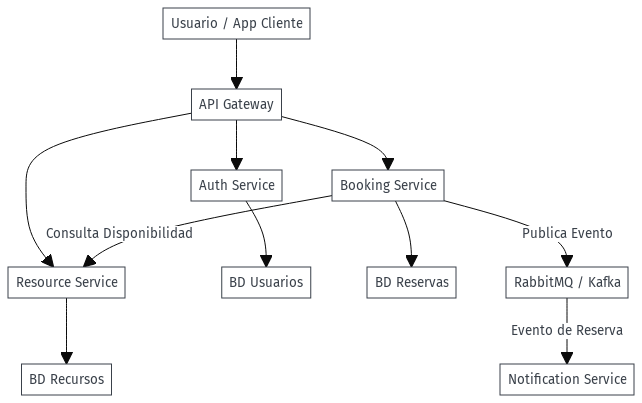
### 2. Arquitectura General (Microservicios)

Se propone una arquitectura distribuida compuesta por los siguientes microservicios, cada uno con una responsabilidad única y desplegado de forma independiente en contenedores Docker.

* **API Gateway:** Punto de entrada único para todas las solicitudes de los clientes. Se encargará del enrutamiento, la autenticación inicial, el balanceo de carga y la limitación de velocidad (rate limiting).
* **Servicio de Autenticación y Usuarios (Auth Service):** Gestiona el registro, inicio de sesión y perfiles de los usuarios. Emite tokens (JWT) para autorizar las solicitudes a otros servicios.
* **Servicio de Recursos (Resource Service):** Administra la información de los espacios y recursos disponibles para reservar (ej. salas de reuniones, equipos, vehículos). Proporciona operaciones CRUD (Crear, Leer, Actualizar, Borrar) para los recursos.
* **Servicio de Reservas (Booking Service):** Contiene la lógica de negocio principal. Gestiona la creación, consulta, modificación y cancelación de reservas. Se comunica con el Servicio de Recursos para verificar la disponibilidad.
* **Servicio de Notificaciones (Notification Service):** Envía comunicaciones a los usuarios, como confirmaciones de reserva, recordatorios o cancelaciones, a través de email o SMS. Opera de forma asíncrona.

### 3. Diagrama de Arquitectura

El siguiente diagrama ilustra el flujo de comunicación entre los componentes del sistema.



**Flujo de Interacción:**

1. El **Usuario** realiza una solicitud a través del **API Gateway**.
2. El **API Gateway** valida el token de autenticación con el **Auth Service**.
3. Una vez autenticado, el Gateway enruta la solicitud al servicio correspondiente (ej. **Booking Service** para crear una reserva).
4. El **Booking Service** se comunica con el **Resource Service** para consultar la disponibilidad del espacio/recurso.
5. Si la reserva es válida, el **Booking Service** la guarda en su base de datos y publica un evento (ej. reserva\_creada) en el **Bus de Mensajería**.
6. El **Notification Service** está suscrito a estos eventos y, al recibirlo, envía una confirmación al usuario.

### 4. Tecnologías Propuestas

* **Lenguaje/Framework Backend:** Java con Spring Boot (robusto, ecosistema maduro y excelente para microservicios).
* **Bases de Datos:**
  + **PostgreSQL:** Para los servicios que requieren consistencia y transacciones (Usuarios, Recursos, Reservas).
  + **Redis:** Para caché de sesiones y datos de acceso frecuente, mejorando el rendimiento.
* **Contenerización:** Docker y Docker Compose para el entorno de desarrollo local.
* **Orquestación:** Kubernetes (K8s) para el despliegue, escalado automático y gestión en producción.
* **Comunicación Asíncrona:** RabbitMQ o Apache Kafka para el bus de eventos entre servicios.
* **API Gateway:** Spring Cloud Gateway o un servicio gestionado por el proveedor cloud (ej. AWS API Gateway).

### 5. Estrategia de Seguridad

* **Autenticación y Autorización:** Se implementará OAuth 2.0 con tokens JWT (JSON Web Tokens). El Auth Service generará los tokens, y el API Gateway los validará en cada solicitud.
* **Encriptación:**
  + **En tránsito:** Se usará HTTPS (TLS) en todos los puntos de comunicación.
  + **En reposo:** Las bases de datos y los volúmenes de almacenamiento estarán encriptados.
* **Gestión de Secretos:** Las credenciales y claves de API se gestionarán a través de un servicio de gestión de secretos como HashiCorp Vault o el servicio equivalente del proveedor de la nube.

### 6. Implementación y Despliegue (CI/CD)

Se propone un pipeline de Integración Continua y Despliegue Continuo (CI/CD) utilizando herramientas como Jenkins o GitLab CI. Cada microservicio tendrá su propio pipeline que incluirá:

1. Compilación del código.
2. Ejecución de pruebas unitarias y de integración.
3. Construcción de la imagen Docker.
4. Publicación de la imagen en un registro de contenedores (ej. Docker Hub, AWS ECR).
5. Despliegue automático en el entorno de Kubernetes correspondiente (desarrollo, staging, producción).