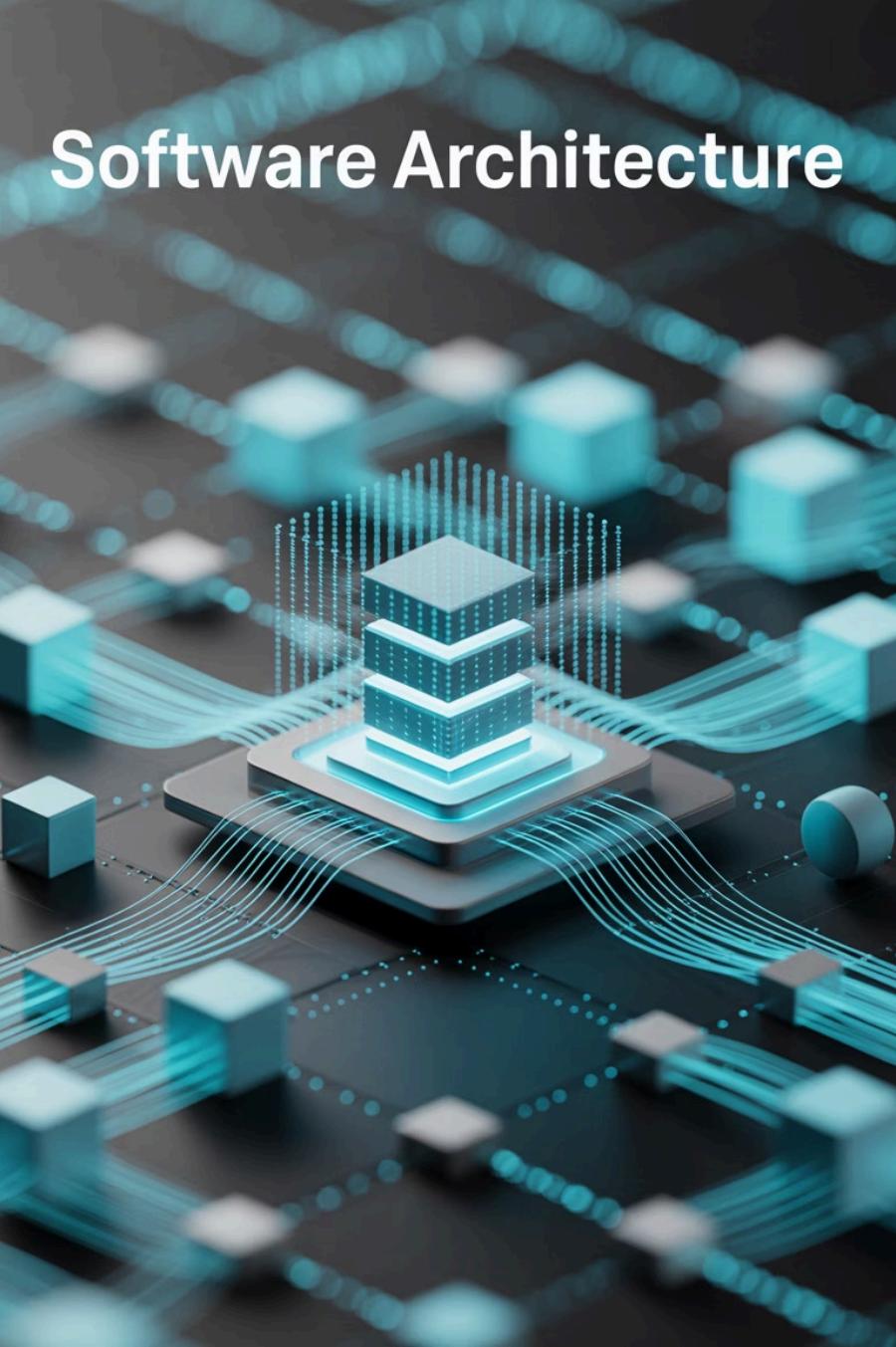


Software Architecture



ARQUITECTURA DE SOFTWARE



Presentación Autores: Andrés David Pérez Cely, Daniel Fernando González Cortés, Juan Diego Reyes Rodríguez Profesor: Andrés Armando Sánchez Martín Bogotá D.C.

Stack Tecnológico

Frontend: VueJS

Backend: FastAPI

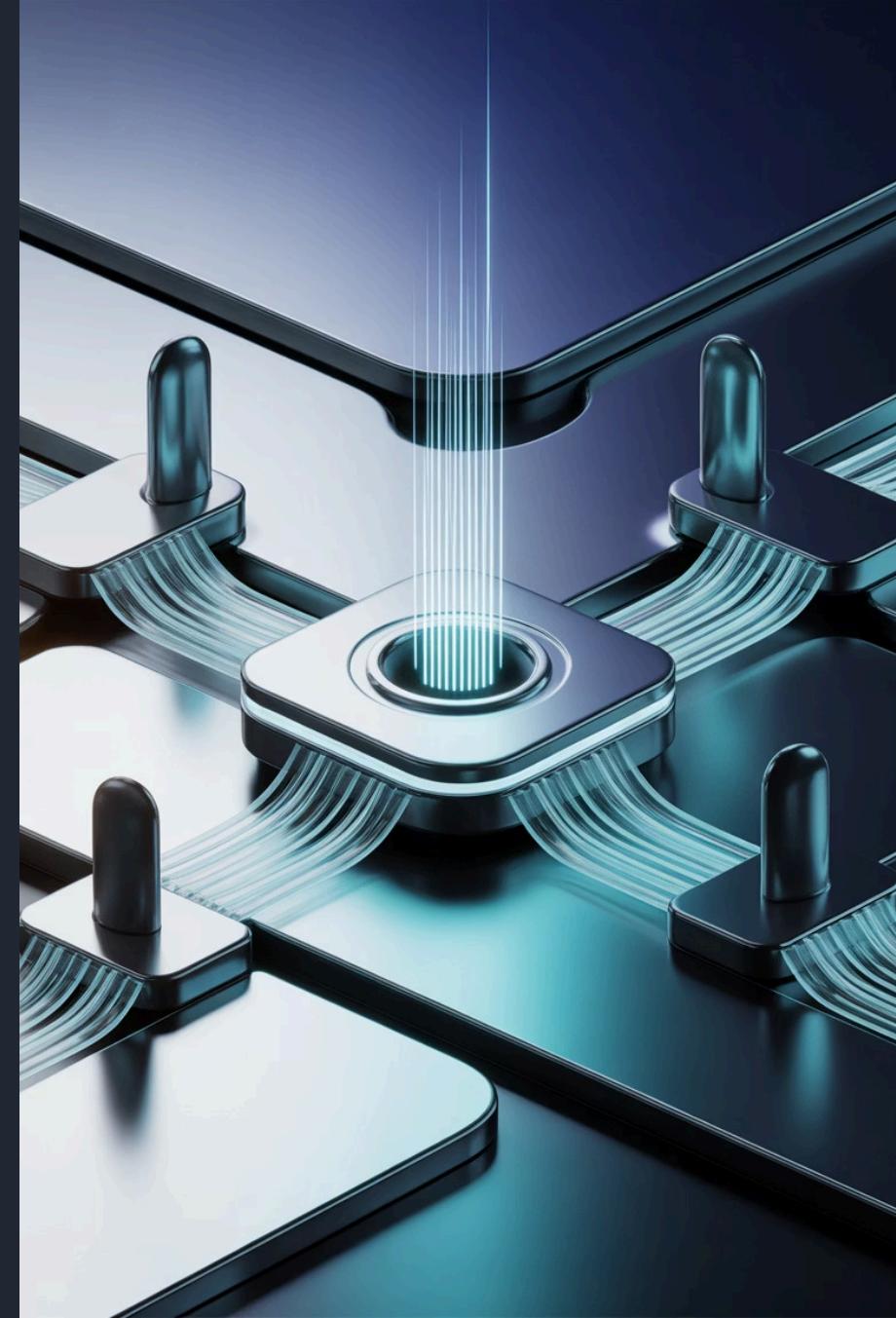
Base de Datos: MySQL

Broker: Mosquitto (MQTT)

Patrón Arquitectónico: Publicador /
Suscriptor (Pub/Sub)



Patrón Publicador / Suscriptor (Publisher–Subscriber)



Definición

Comunicación Asíncrona

Modelo de **comunicación asíncrona y desacoplada** entre componentes.

Publishers y Subscribers

Publishers emiten eventos a *topics*; **subscribers** reciben lo que les interesa.

Broker Central

Un **broker** (p. ej., MQTT) **distribuye** los mensajes.

Características



Desacoplamiento

Emisores y receptores no se conocen.



Basado en eventos

Flujo continuo de notificaciones.



Fiable

QoS, persistencia, sesiones (según protocolo).



Asincronía

No requiere respuesta inmediata.



Escalable y flexible

Múltiples publishers/subscribers.



Multiprotocolo

MQTT, Kafka, RabbitMQ, Redis, etc.

Rol del Broker en el Patrón Publisher–Subscriber

El **broker** (intermediario o gestor de mensajes) es el **núcleo** del patrón **Pub/Sub**. Su función es **recibir, filtrar y distribuir** los mensajes entre los **publicadores (publishers)** y los **suscriptores (subscribers)** según los *topics* o temas de interés.

Funciones principales del broker

Recepción de mensajes

recibe las publicaciones que envían los publicadores.

Gestión de tópicos

organiza los mensajes por temas (*topics*) para que solo los suscriptores interesados los reciban.

Entrega garantizada

aplica reglas de calidad de servicio (QoS) para asegurar que los mensajes lleguen correctamente.

Almacenamiento temporal

puede retener mensajes para suscriptores desconectados (según la configuración del protocolo).

Seguridad

maneja autenticación, autorización y cifrado para evitar accesos no autorizados.

Monitoreo y control

registra métricas sobre conexión, latencia y tráfico de mensajes.

Niveles de Calidad de Servicio (QoS) en MQTT

El protocolo **MQTT** incluye tres niveles de **Quality of Service (QoS)** que determinan cómo se garantiza la entrega de los mensajes entre el publicador, el broker y el suscriptor. Esto permite ajustar el equilibrio entre **rendimiento, confiabilidad y consumo de red** según el caso de uso.

Nivel	Nombre	Descripción	Uso recomendado
QoS 0	<i>At most once</i> (como máximo una vez)	El mensaje se envía sin confirmación. Si se pierde, no se reintenta. Es el modo más rápido, pero menos confiable.	Telemetría no crítica, sensores frecuentes.
QoS 1	<i>At least once</i> (al menos una vez)	El publicador espera confirmación del broker. Puede llegar más de una vez si hay retransmisión.	Comunicación general o eventos donde duplicados no sean un problema.
QoS 2	<i>Exactly once</i> (exactamente una vez)	Intercambio de cuatro pasos entre publicador y broker para garantizar una única entrega sin duplicados. Es el más confiable pero también el más lento.	Transacciones críticas o datos financieros.

Suscripción y temas (Topics)

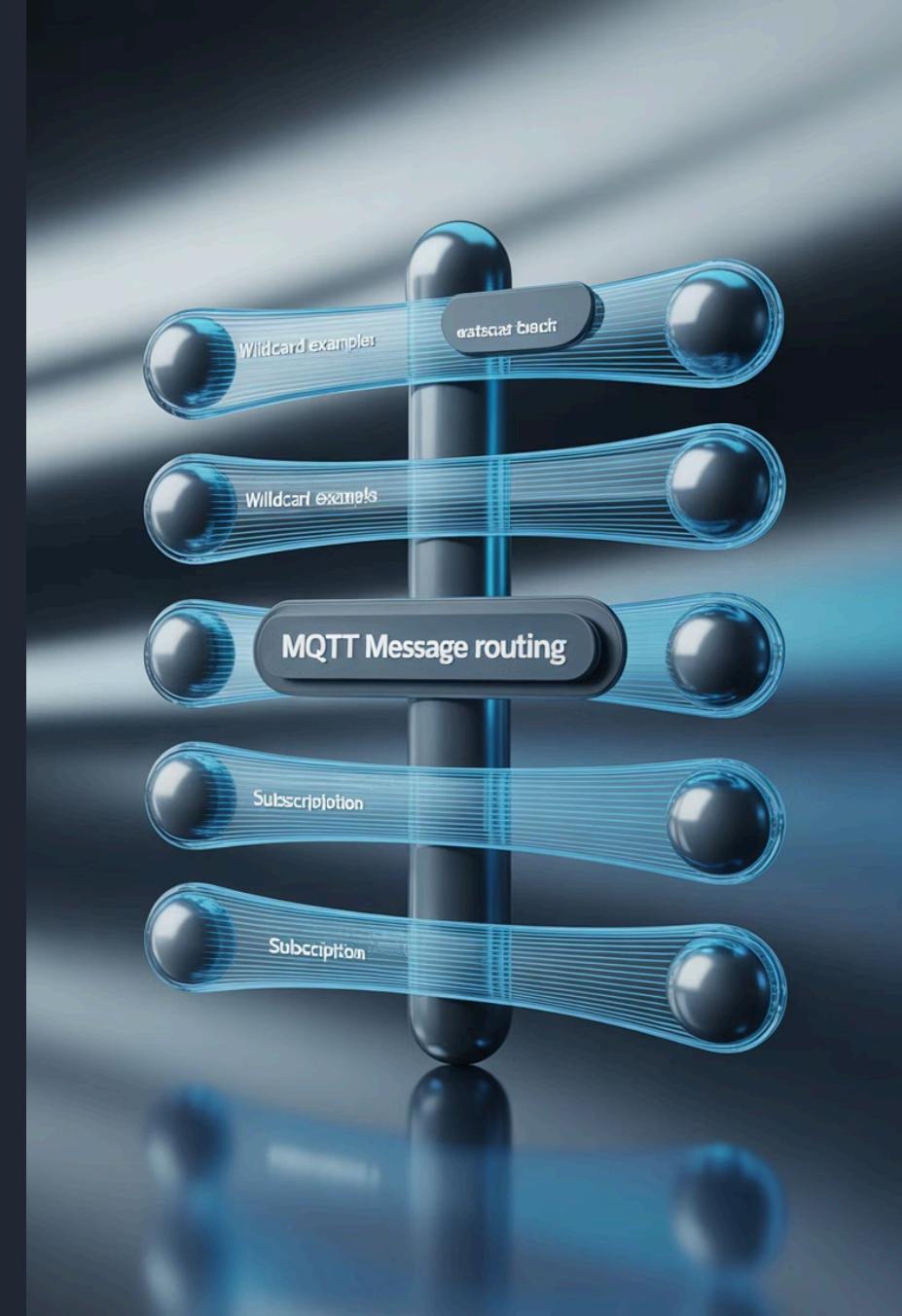
Un **topic** es el **canal lógico** por el que se enrutan los mensajes en **Pub/Sub (MQTT)**. Los **publicadores** envían mensajes a un *topic* y los **suscriptores** reciben solo los de los *topics* a los que están suscritos.

Jerarquía y sintaxis

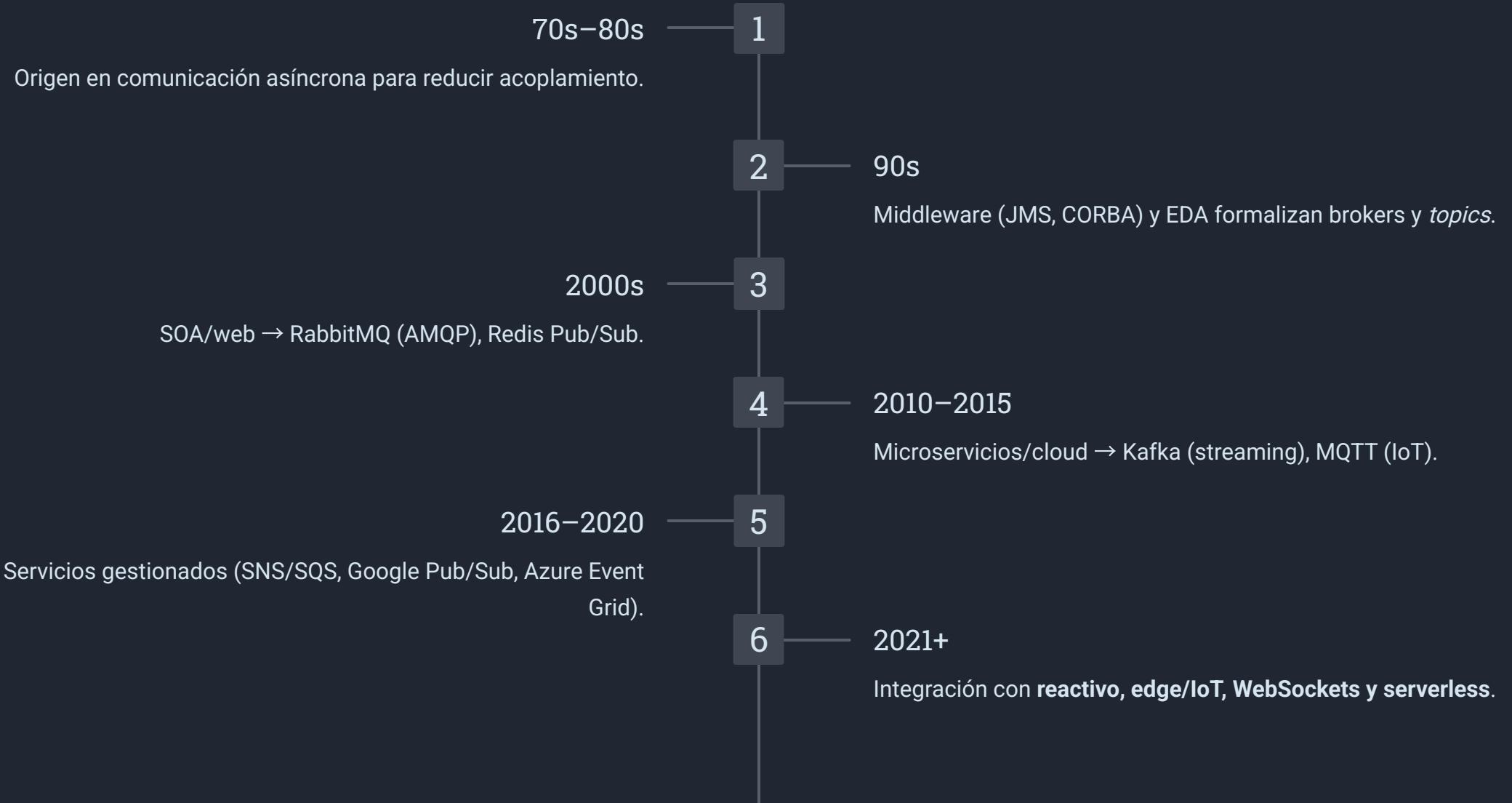
- Los *topics* se organizan **jerárquicamente** con / como separador.
- No hay registro previo: un *topic* existe cuando se publica o se suscribe.
- **Ejemplos:**
- noticias/publicadas
- noticias/categoría/política
- usuarios/123/actividad

Comodines (wildcards)

- + (**un nivel**): coincide con **un** segmento.
- noticias/categoría/+ → recibe noticias/categoría/política y noticias/categoría/deportes.
- # (**varios niveles**): coincide con **cero o más** segmentos al final.
- noticias/# → recibe **todo** bajo noticias/...
- **Reglas:** los wildcards solo se usan **al suscribirse**, no al publicar.



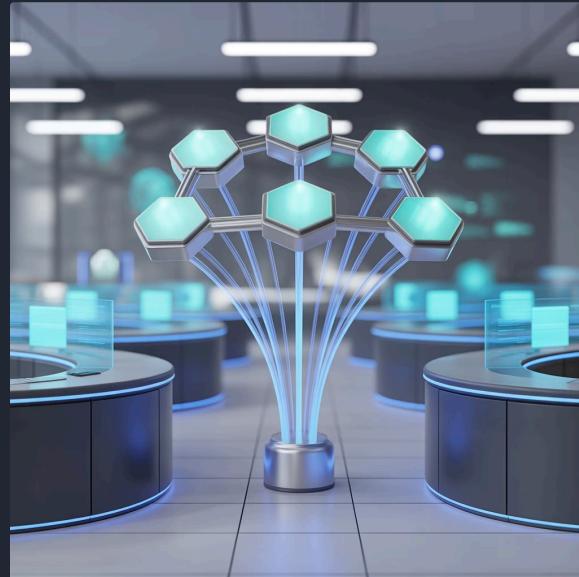
Historia y evolución



Ventajas y Desventajas

Ventajas

- Desacoplamiento, escalabilidad, asincronía.
- Flexibilidad tecnológica vía broker.
- Tolerancia a fallos y soporte tiempo real.
- Mejor mantenibilidad (módulos claros).



Desventajas

- Más complejidad operativa (broker).
- Trazabilidad y depuración más difíciles.
- Riesgo de mensajes perdidos si QoS mal configurado.
- Dependencia del broker y posible sobrecarga de red.
- Requiere seguridad (auth/TLS/ACL).

Casos de Uso

Situaciones y/o problemas donde se pueden aplicar

Notificaciones en tiempo real

Situación: Usuarios deben recibir nuevas publicaciones sin recargar la página.

Aplicación: El servidor publica eventos y los clientes suscritos los reciben mediante WebSockets o MQTT.

Internet de las Cosas (IoT)

Situación: Sensores distribuidos envían datos periódicamente a un sistema central.

Aplicación: Los sensores publican datos en *topics*, y los sistemas de monitoreo se suscriben solo a los relevantes.

Monitoreo y telemetría

Situación: Microservicios necesitan reportar métricas o fallos en tiempo real.

Aplicación: Cada servicio publica eventos de estado y herramientas como Grafana o Prometheus se suscriben.

Más Casos de Uso

Situaciones y/o problemas donde se pueden aplicar

Aplicaciones financieras o de trading

Situación: Las cotizaciones cambian constantemente.

Aplicación: Servicios publican precios y los clientes suscritos reciben actualizaciones instantáneas.

Comunicación entre microservicios

Situación: Múltiples módulos deben reaccionar ante eventos sin acoplamiento.

Aplicación: Cada servicio publica eventos ("usuario/creado") y otros se suscriben.

Control industrial y automatización

Situación: Equipos deben coordinar operaciones en tiempo real.

Aplicación: Controladores publican estados y comandos en canales MQTT.

Chats y mensajería instantánea:

Situación: Usuarios deben recibir mensajes nuevos en tiempo real.

Aplicación: Cada sala de chat funciona como un *topic* al que los usuarios están suscritos.

Casos de aplicación

Ejemplos y casos de éxito en la industria



Netflix – Microservicios y mensajería interna

Utiliza Pub/Sub para comunicación asíncrona entre servicios.

Resultado: Mayor resiliencia y escalabilidad global.



Uber – Eventos entre servicios

Implementa Kafka para coordinar viajes, precios y notificaciones.

Resultado: Sincronización en tiempo real y alta disponibilidad.



Meta (Facebook) – Actividad en tiempo real

Usa Pub/Sub para distribuir likes, comentarios y mensajes.

Resultado: Experiencia fluida y de baja latencia.



Amazon Web Services (AWS) – SNS y SQS

Servicios administrados basados en Pub/Sub.

Resultado: Reducción de costos y facilidad de integración.



Google Cloud – Pub/Sub como servicio

Maneja millones de eventos por segundo.

Resultado: Procesamiento en streaming y analítica en tiempo real.



Tesla – Telemetría vehicular

Usa MQTT y Pub/Sub para enviar datos de vehículos.

Resultado: Monitoreo continuo y actualizaciones OTA.



Spotify – Sincronización entre dispositivos

Aplica Pub/Sub para mantener reproducción simultánea.

Resultado: Experiencia sincronizada y estable.

Industria de IoT – Manufactura y automatización

Empresas como Siemens, Schneider y Bosch emplean MQTT con Pub/Sub.

Resultado: Comunicación confiable y control remoto eficiente.

El patrón Publisher–Subscriber ha revolucionado la arquitectura de software moderna, permitiendo sistemas más escalables, flexibles y resilientes que pueden adaptarse a las demandas del mundo digital actual.



Vue.js – Definición

- Framework JavaScript para **interfaces web interactivas y dinámicas**
- **Progresivo**: fácil de adoptar desde proyectos pequeños a complejos
- **Basado en componentes** y reactividad automática de la UI

Vue.js – Características

Facilidad de uso

Sintaxis clara

Reactividad

La UI se actualiza con los datos

Componentes

Reutilizables

- **Integración progresiva** en proyectos existentes
- **Data binding** y ecosistema: **Vue Router, Pinia/Vuex**
- **Rendimiento** y carga ligera

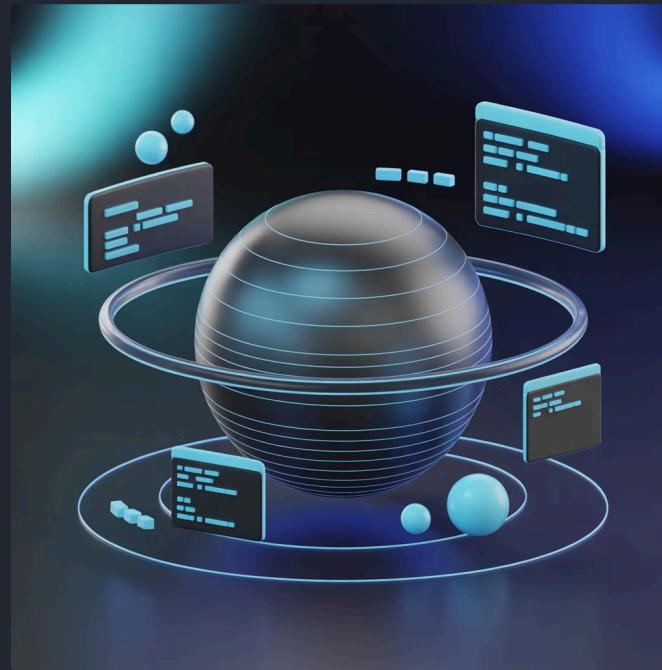
Vue.js – Historia y evolución



Vue.js – Ventajas y Desventajas

Ventajas

- Fácil de aprender, alta reactividad, componentes reutilizables
- Buen rendimiento, excelente documentación, comunidad activa



Desventajas

- Menor respaldo corporativo
- Ecosistema fragmentado (librerías no oficiales)
- Complejidad en proyectos muy grandes y compatibilidades entre versiones
- Menor demanda relativa vs. React/Angular

Vue.js – Casos de uso

SPAs

Paneles, formularios, dashboards en tiempo real

Integración progresiva

En sitios existentes

E-commerce

Catálogos, carritos, búsqueda rápida

Apps móviles/multiplataforma

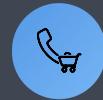
Con Quasar o NativeScript-Vue



Vue.js – Casos de aplicación (Industria)



Alibaba - Usa Vue.js para interfaces ligeras y rápidas que escalan con millones de productos.



Xiaomi - Implementa Vue.js en su web/tiendas para una compra ágil y adaptable en distintos dispositivos.



Nintendo - Emplea Vue.js en portales con contenidos y promociones de alto dinamismo visual.



Behance - Utiliza Vue.js para cargar proyectos de forma dinámica y mejorar el rendimiento del sitio.



GitLab - Integra Vue.js en la UI para una experiencia fluida en gestión de repos, issues y reviews.



Grammarly - Integra Vue.js en la UI para una experiencia fluida en gestión de repos, issues y reviews.



FastAPI – Definición

- Framework web moderno y rápido en Python para APIs
- Construido sobre Starlette (web/async) y Pydantic (validación)
- Soporta ASGI (async/await), genera OpenAPI, Swagger UI y ReDoc

FastAPI – Características



Alto rendimiento

(ASGI, Uvicorn)



Validación automática

Con Pydantic



Docs interactivas

Auto-generadas



Type hints

DX, autocompletado, menos errores



Compatibilidad con estándares

(OpenAPI, JSON Schema)

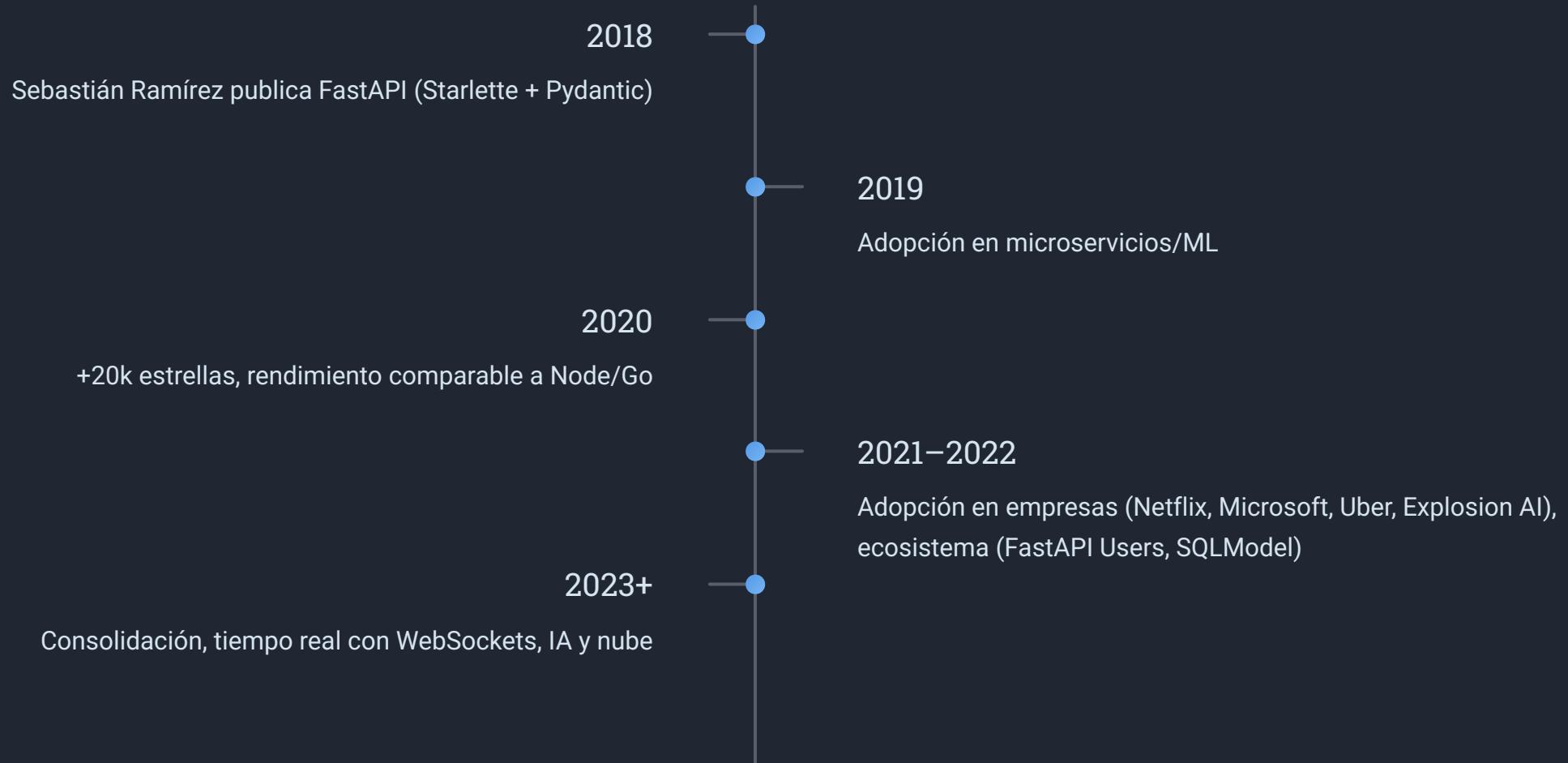


Seguridad

(OAuth2, JWT)

Escalable, integración fácil con DBs, colas, Docker/K8s

FastAPI – Historia y evolución



FastAPI – Ventajas y Desventajas

Ventajas

- Rendimiento
- Validación automática, serialización automática
- Documentación integrada con OpenAPI y JSON Schema
- Programación asíncrona: Async/await
- Tipado estático, integración con múltiples herramientas



Desventajas

- Menos maduro que Django/Flask
- Menos tutoriales avanzados
- Curva de aprendizaje compleja por type hints
- Complejidad por async en apps simples
- Autenticación/sesiones deben integrarse

FastAPI – Casos de uso



Microservicios

Y APIs empresariales



MLOps

Servir modelos de IA



Tiempo real

WebSockets, alta concurrencia



API Gateway

E integraciones externas



Mobile/web backends

Con JWT/OAuth2



IoT

Con MQTT/WebSockets



Prototipado

Veloz con docs automáticas

FastAPI – Casos de aplicación (Industria)

Netflix

Automatización y datos internos para procesamiento y análisis de datos

Microsoft (Azure ML)

Adopción de FastAPI en Azure Machine Learning para desplegar modelos de IA como servicios REST

Uber

Microservicios relacionados con su infraestructura de datos.

Explosion AI (spaCy/Prodigy)

Backend NLP

DocuBrain

startup que implementó FastAPI como backend para un sistema de análisis inteligente de documentos.

The MySQL logo is displayed as a watermark in the background of the slide. It consists of the word "MySQL" in a stylized, italicized font, with a blue and white color scheme. The letters are partially obscured by a grid of light blue and orange lines that radiate from the top left corner, creating a sense of motion and depth.

MySQL – Definición

- **Sistema de gestión de bases de datos relacional** (RDBMS)
- **SQL** estándar; **open source**, rápido, estable y popular
- Amplia **compatibilidad** con lenguajes y plataformas

MySQL – Características

SQL estándar

Modelo relacional

Alto rendimiento

Seguridad y control de acceso

Multiplataforma

Código abierto

Transacciones

(Integridad) y escalabilidad

MySQL – Historia y evolución

- 
- 1994–1995
Creación (Widenius/Axmark), primera versión pública
 - 2000
GPL → crecimiento (pilar del stack **LAMP**)
 - 2005
MySQL AB comprada por **Sun**
 - 2010
Oracle adquiere Sun (Community vs Enterprise)
 - 2013–2019
Mejoras y **MariaDB** como derivado
 - 2020–2025
Cloud (**HeatWave**), más seguridad y rendimiento

MySQL – Ventajas y Desventajas

Ventajas

- Gratuito/open source
- Rendimiento y velocidad
- Multiplataforma
- Fácil de usar
- Seguro
- Gran comunidad



Desventajas

- Límites con consultas muy pesadas / datos extremadamente grandes
- Funcionalidades Enterprise de pago
- Dependencia de Oracle
- Menos herramientas gráficas integradas
- Competencia de PostgreSQL/NoSQL en entornos muy grandes

MySQL – Casos de uso



Sitios web dinámicos

(Usuarios, productos, comentarios)



Aplicaciones empresariales

(Ventas, nómina, inventario)



Educación

(Plataformas académicas, bibliotecas)



Reservas y logística



Apps móviles

Y servicios en la nube



Análisis y reporting

MySQL – Casos de aplicación (Industria)

Facebook

Usa MySQL para datos de usuarios, publicaciones y relaciones, soportando millones de consultas por segundo

Shopify

MySQL maneja millones de transacciones y catálogos en miles de tiendas con alta fiabilidad y escala.

Airbnb

MySQL sustenta reservas, usuarios y propiedades globales, garantizando coherencia y disponibilidad.



YouTube

Gestionó videos, usuarios y comentarios en MySQL, clave para su crecimiento temprano por rendimiento y estabilidad

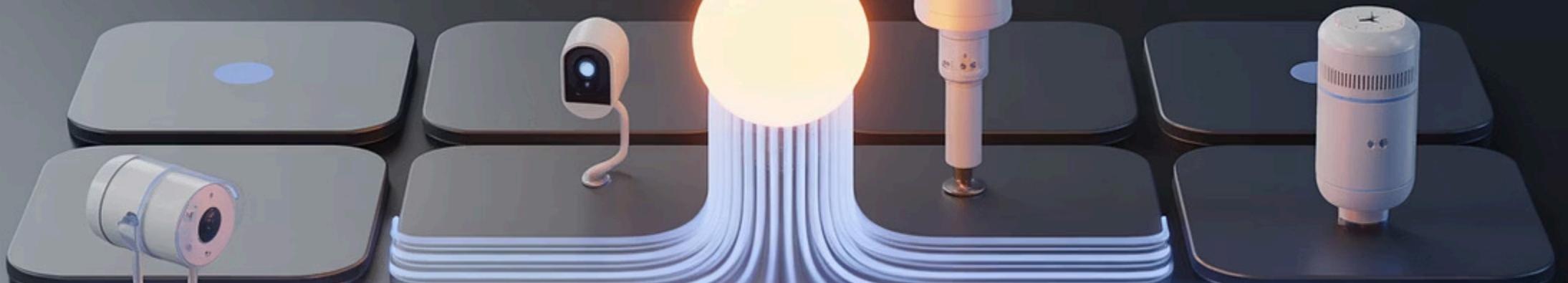
Twitter

Empleó MySQL en sus inicios para manejar millones de tuits y perfiles durante la expansión.

Netflix

Utiliza MySQL para datos críticos (usuarios, dispositivos, preferencias) integrado con otras tecnologías.

Motivos: **rendimiento, estabilidad, escala**



MQTT – Definición

Protocolo ligero

Protocolo ligero **pub/sub** sobre **TCP/IP**

Optimización

Optimizado para **IoT**, **telemetría** y **sistemas embebidos**

Arquitectura

Broker intermedia entrega; soporta **QoS** y conexiones persistentes

MQTT – Características



Pub/Sub con broker

Pub/Sub con broker (topics)



Ligero y eficiente

Ligero y eficiente (header 2 bytes)



Conexión persistente

Conexión persistente



Quality of Service

QoS 0/1/2



Mensajes persistentes

Retained/persistent messages, sesiones



Seguridad

Seguridad: auth, ACL, **TLS/SSL**



Integración

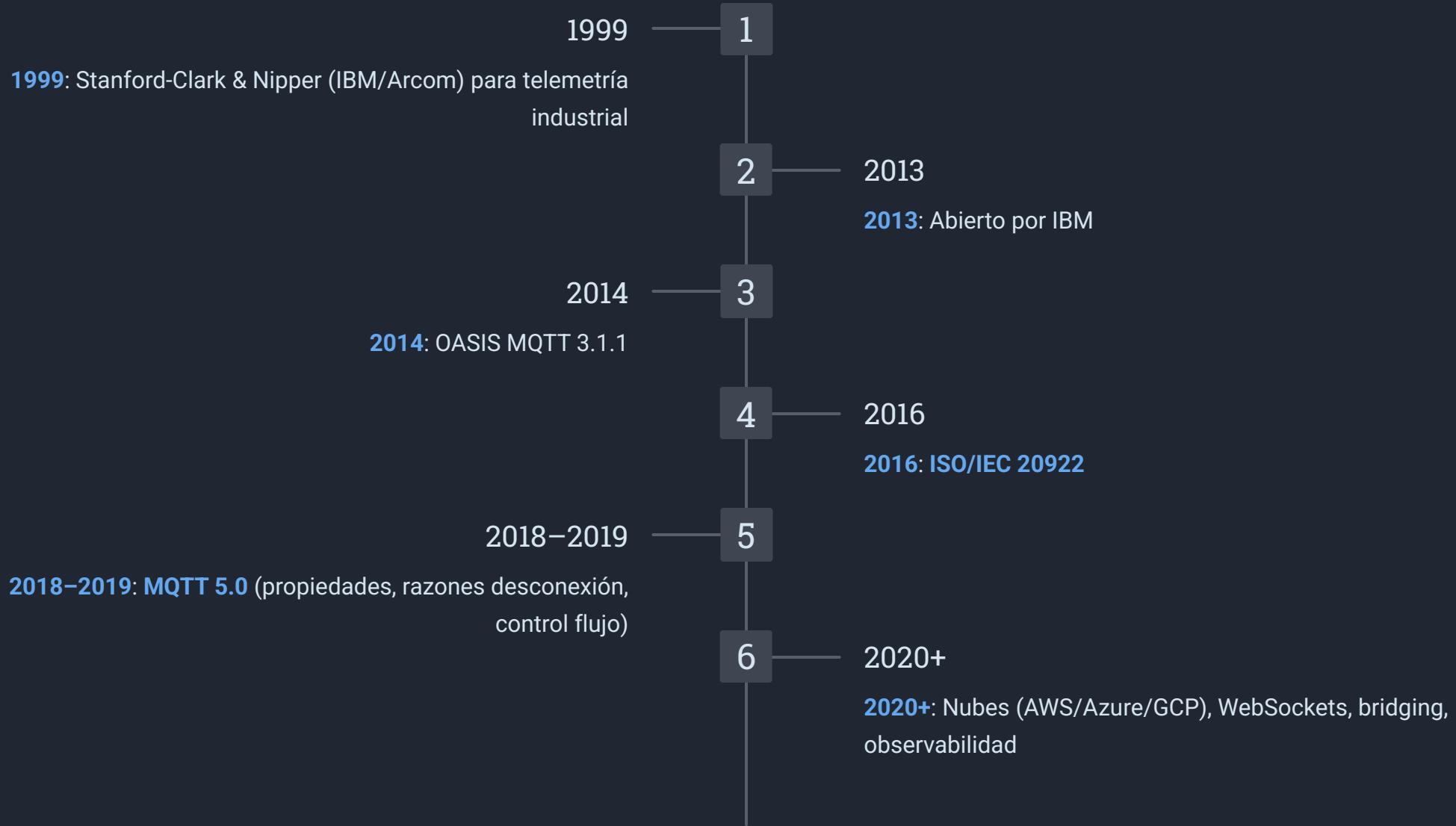
Integración con HTTP/WebSockets/AMQP



Estándar abierto

Estándar abierto ISO/IEC 20922

MQTT – Historia y evolución





MQTT – Ventajas y Desventajas

Ventajas

- **Ligereza, eficiencia, QoS, persistencia, escalabilidad**
- **Seguridad TLS/ACL, interoperabilidad multi-lenguaje**

Desventajas

- **Dependencia del broker**
- Requiere **conexión TCP persistente**
- **Seguridad** no viene activada por defecto (configurar TLS)
- No apto para **grandes volúmenes**/multimedia

Niveles de Calidad de Servicio (QoS) en MQTT

Nivel	Nombre	Descripción	Uso recomendado
QoS 0	<i>At most once</i> (como máximo una vez)	El mensaje se envía sin confirmación. Si se pierde, no se reintenta. Es el modo más rápido, pero menos confiable.	Telemetría no crítica, sensores frecuentes.
QoS 1	<i>At least once</i> (al menos una vez)	El publicador espera confirmación del broker. Puede llegar más de una vez si hay retransmisión.	Comunicación general o eventos donde duplicados no sean un problema.
QoS 2	<i>Exactly once</i> (exactamente una vez)	Intercambio de cuatro pasos entre publicador y broker para garantizar una única entrega sin duplicados. Es el más confiable pero también el más lento.	Transacciones críticas o datos financieros.

MQTT – Casos de uso



IoT

IoT (sensores, dashboards en tiempo real)



Automatización industrial

Automatización industrial/SCADA



Domótica

Domótica



Vehículos conectados

Vehículos conectados



Smart Grids

Smart Grids



Salud digital

Salud digital



Agricultura inteligente

Agricultura inteligente



Notificaciones

Notificaciones en tiempo real

MQTT – Casos de aplicación (Industria)

Facebook Messenger

Facebook Messenger: adoptó MQTT para mensajería móvil eficiente

Cloud Platforms

AWS IoT Core, Azure IoT Hub: comunicación dispositivo-nube

Volkswagen

Volkswagen: vehículos conectados (diagnóstico/mantenimiento remoto)

Siemens

Siemens: soluciones IIoT y sistemas SCADA

Telecomunicaciones

Ericsson/Telefónica: Smart Cities

Google Nest

Google Nest: domótica, hogares inteligentes

Agriculture

John Deere / Agri-Tech: agricultura conectada

¿Qué tan común es el stack?

Sí, bastante, sobre todo en IoT y dashboards

En web general hay alternativas populares, pero la combinación **Vue.js** + **FastAPI** + **MySQL** + **MQTT** es sólida y práctica

Code Elevated



VUE.JS



FastAPI



MySQL

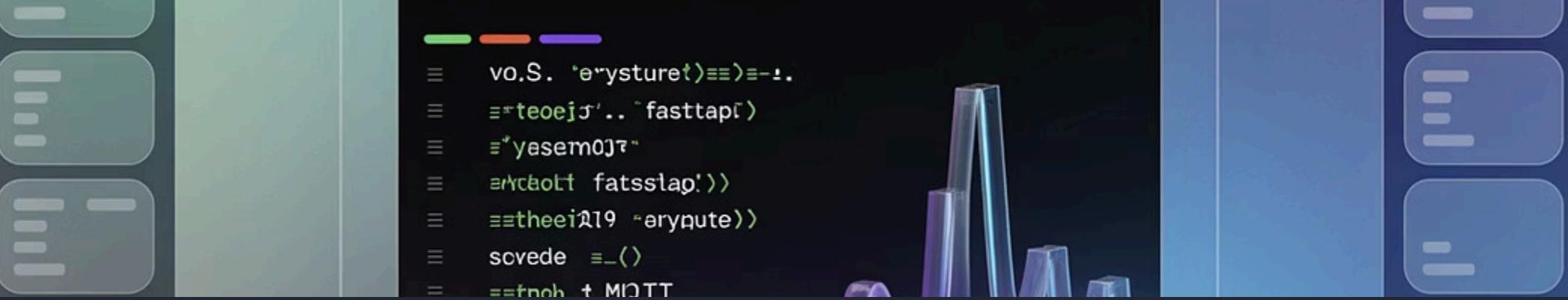


MQTT

Agile Development

Cloud integration

Scalable Architectures



¿Por qué usar este stack?



Vue.js

Vue.js: rápido de montar, ideal para **SPA** y **dashboards**



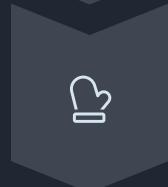
FastAPI

FastAPI: APIs Python **modernas, tipadas y rápidas**



MySQL

MySQL: **estable** y muy usado en producción



MQTT

MQTT: estándar de facto para IoT/telemetría

¿Qué tan comunes son cada uno?



MySQL

Muy común en todo tipo de proyectos



Vue.js

Común, especialmente en dashboards
(React domina cuota global)



FastAPI

En crecimiento para APIs nuevas en
Python



MQTT

Muy común en IoT (menos fuera de IoT)

Technology Adoption rates



Mercado laboral por tecnología

Resumen orientativo a nivel global:

Vue.js (Frontend)

Demandas altas; fortalezas en SPA y DX; desafío frente a React; salarios competitivos en frontend moderno.

FastAPI (Backend)

En crecimiento; fortalezas en rendimiento y tipado; desafío por menor madurez vs Django/Flask; buenas oportunidades en data/IA.

MySQL (Base de Datos)

Demandas muy altas; fortalezas en estabilidad y facilidad; compite con PostgreSQL/NoSQL en big data; roles de DBA/Backend.

MQTT (IoT)

Alta en IoT/IIoT; fortalezas en eficiencia y tiempo real; menor demanda fuera de IoT; oportunidades en edge/cloud y telecom.



Vue.js (Frontend)

Demanda laboral: Alta — muy solicitado en el desarrollo frontend. **Fortalezas:** Framework moderno, excelente para SPA, comunidad activa. **Desafíos / competencia:** Alta competencia con React; aún existen proyectos en Vue 2.

\$114,785

Anual

\$9,565

Mensual

\$441

Diario

Nivel típico: Intermedio / Senior



FastAPI (Backend con Python)

Demanda laboral: En crecimiento — preferido por startups y empresas de IA. **Fortalezas:** Alto rendimiento, tipado estático, documentación automática, API modernas. **Desafíos / competencia:** Ecosistema más joven frente a Django o Flask.

\$109,905

Anual

\$9,159

Mensual

\$423

Diario

Nivel típico: Intermedio / Senior



MySQL (Base de datos relacional)

Demanda laboral: Muy alta – estándar en el entorno empresarial. **Fortalezas:** Estabilidad, confiabilidad, soporte masivo y multiplataforma. **Desafíos / competencia:** PostgreSQL gana terreno por sus capacidades avanzadas.

\$46,679

Anual

\$3,889

Mensual

\$180

Diario

Nivel típico: Junior / Intermedio





MQTT (Mensajería e Integración IoT)

Demanda laboral: Especializada, pero en expansión gracias al auge del IoT. **Fortalezas:** Comunicación ligera, ideal para dispositivos con recursos limitados. **Desafíos / competencia:** Poca oferta de talento especializado y menor demanda fuera del IoT.

\$147,514

Anual

\$12,293

Mensual

\$567

Diario

Nivel típico: Intermedio / Senior

Principio SOLID

Principio	Vue.js (Frontend)	FastAPI (Backend)	MySQL (Base de Datos)	MQTT (Integración)
S – Responsabilidad Única	Cada componente maneja una función específica (formulario, lista de noticias, etc.).	Cada endpoint se encarga de una sola operación (crear noticia, obtener suscriptores, etc.).	Cada tabla almacena datos de una sola entidad (usuarios, noticias, mensajes).	Cada tópico MQTT se dedica a un tipo de información (por ejemplo, noticias/nacionales).
O – Abierto/Cerrado	Se pueden agregar nuevos componentes sin modificar los existentes.	Se pueden extender servicios agregando nuevas rutas sin alterar las ya creadas.	Es posible añadir nuevas tablas o campos sin afectar las consultas principales.	Se pueden crear nuevos tópicos o suscriptores sin modificar los anteriores.
L – Sustitución de Liskov	Componentes derivados mantienen la estructura y comportamiento base (por ejemplo, un CardNoticia hereda de CardBase).	Servicios o clases hijas respetan la interfaz esperada de las clases base (por ejemplo, PublicadorService y SuscriptorService).	Vistas o relaciones entre tablas pueden sustituirse sin romper consultas.	Un nuevo cliente MQTT puede reemplazar otro mientras siga las reglas del protocolo.
I – Segregación de Interfaces	Los componentes solo reciben las props que necesitan (no se cargan funciones innecesarias).	Las rutas de la API están divididas por módulos (por ejemplo, /noticias, /usuarios, /suscriptores).	Cada consulta o vista solo devuelve los campos necesarios.	Los suscriptores se conectan únicamente a los tópicos que les interesan.
D – Inversión de Dependencias	Los componentes dependen de datos externos (API) y no directamente del backend.	FastAPI usa inyección de dependencias (por ejemplo, el servicio MQTT o la conexión a BD).	El acceso a datos se hace mediante un ORM (SQLAlchemy), no consultas directas.	Los servicios publican y reciben mensajes a través del broker, no de forma directa entre ellos.

Atributos de calidad (ISO/IEC 25010)

Atributo	VueJS	FastAPI	MySQL	MQTT
Adecuación funcional	UI reactiva; facilita cumplir requerimientos funcionales en el cliente.	Validación y serialización con Pydantic; OpenAPI automático.	ACID; SQL rico; vistas/funciones/triggers para reglas de negocio.	Pub/Sub ligero; entrega por QoS; retained/last will.
Eficiencia de desempeño	DOM virtual y reactividad eficiente; buen rendimiento en SPA.	ASGI + async/await; muy baja latencia; alto throughput.	Índices, caché de consultas; buen rendimiento OLTP.	Encabezado mínimo; excelente en enlaces de baja banda.
Compatibilidad	Integra con REST/WebSockets; convive con librerías externas.	Interopera vía OpenAPI; fácil orquestación con microservicios.	Conectores amplios; replica y federa con otras BD.	Interopera con HTTP/WebSockets/bridges entre brokers.
Usabilidad	Curva amable, DX sólida; ecosistema de componentes.	Docs interactivas (Swagger/ReDoc); DX excelente.	Herramientas maduras (Workbench); lenguaje estándar SQL.	Modelo simple de topics; fácil de adoptar en IoT.
Confiabilidad	Depende del navegador; pruebas E2E/unitarias mejoran estabilidad.	Starlette estable; pruebas con TestClient; manejo de errores estructurado.	Replicación, backups, InnoDB; alta disponibilidad con clusters.	Sesiones persistentes; reintentos vía broker; tolerante a cortes.
Seguridad	XSS/CSRF mitigables con buenas prácticas; requiere hardening.	OAuth2/JWT, dependencias, TLS vía proxy; fácil de reforzar.	Cifrado en tránsito/descanso; usuarios/roles/GRANTs.	TLS/SSL, auth por usuario/ACLs; requiere configuración cuidadosa.
Mantenibilidad	Arquitectura por componentes y tipado opcional (TS) facilitan cambios.	Código tipado; modular; fácil refactor con type hints.	Esquemas versionables; migraciones (Liquibase/Flyway).	Clientes simples; topologías escalables; bajo acoplamiento.
Portabilidad	Funciona en navegadores modernos; SSR/Nuxt para distintos entornos.	Despliegue en Uvicorn/Gunicorn; contenedores; nubes múltiples.	Multiplataforma; contenedores; cloud managed (RDS, etc.).	Corre en múltiples brokers y dispositivos; apto edge/cloud.

Tácticas / Temas

Táctica	VueJS	FastAPI	MySQL	MQTT
Tácticas de rendimiento	DOM virtual, lazy loading, rendering reactivo.	Uso de async/await, alto rendimiento con ASGI y Uvicorn, caché de respuestas.	Uso de índices, optimización de consultas, caché de resultados y buffer pool.	Mensajes ligeros, encabezado mínimo, comunicación asíncrona persistente.
Tácticas de seguridad	Protección contra XSS/CSRF, validación en formularios, uso de HTTPS.	Autenticación con OAuth2 y JWT, cifrado TLS, validación automática con Pydantic.	Roles y permisos, cifrado de datos en tránsito y en reposo, auditorías.	Autenticación por usuario/clave, TLS/SSL, listas de control de acceso (ACL).
Tácticas de disponibilidad	Recuperación del estado del componente y reactividad en tiempo real.	Despliegue en múltiples instancias, soporte de balanceo de carga y recuperación ante fallos.	Replicación, clustering, backups automáticos, recuperación ante fallos.	Sesiones persistentes, QoS, brokers redundantes para alta disponibilidad.
Tácticas de mantenibilidad	Componentes reutilizables y desacoplados; separación lógica de vista y estado.	Código modular, tipado fuerte, dependencias bien definidas.	Normalización de esquemas, scripts de migración y consistencia ACID.	Topología desacoplada, simplicidad de clientes y modularidad del protocolo.
Tácticas de escalabilidad	Carga diferida de componentes, división de código, optimización de recursos.	Arquitectura de microservicios, contenedorización, integración con orquestadores.	Sharding, replicación maestro-esclavo y balanceo de carga de consultas.	Soporte para miles de clientes concurrentes, brokers escalables, bridging entre nodos.
Tácticas de usabilidad	Binding bidireccional, interfaz intuitiva, comunicación fluida con el usuario.	Documentación automática (Swagger/ReDoc), endpoints claros y bien estructurados.	Herramientas gráficas y sintaxis SQL estándar ampliamente conocida.	Modelo simple de topics, facilidad de adopción en entornos IoT y bajo consumo de recursos.

Patrones / Temas

Patrón	VueJS	FastAPI	MySQL	MQTT
Modelo–Vista–Controlador (MVC)	Implementa la Vista y parte del Controlador en el cliente mediante componentes y reactividad.	Cumple el rol de Controlador y parte del Modelo en aplicaciones RESTful.	Representa el Modelo dentro del patrón MVC, almacenando y estructurando los datos.	No aplica al MVC clásico; se centra en la comunicación entre servicios.
Cliente–Servidor	Funciona como cliente que consume servicios REST o GraphQL desde el servidor.	Actúa como servidor que expone endpoints al cliente (VueJS u otros).	Opera como servidor de base de datos que responde a consultas SQL del backend.	Opera como middleware dentro de una arquitectura cliente–servidor extendida.
Microservicios	Consume microservicios a través de endpoints; se integra fácilmente con APIs independientes.	Soporta arquitectura de microservicios mediante APIs independientes y contenedorización.	Cada microservicio puede tener su propia instancia o base dedicada.	Integra microservicios a través del intercambio de mensajes asincrónicos.
Publicador–Suscriptor (Pub/Sub)	Puede suscribirse a canales de datos en tiempo real mediante WebSockets o MQTT.	Puede publicar o suscribirse a mensajes usando brokers MQTT o Redis.	Puede almacenar mensajes o logs generados por sistemas Pub/Sub.	Es la implementación por excelencia del patrón Publicador–Suscriptor.
Repository / DAO	No aplica directamente; consume datos gestionados por el backend.	Implementa repositorios y modelos para la capa de acceso a datos.	Es la base del patrón Repository, centralizando la persistencia de datos.	No aplica.
Event-Driven	Puede reaccionar a eventos de usuario y cambios en datos en tiempo real.	Permite manejar eventos asincrónicos y WebSockets.	Soporta triggers y eventos SQL.	Se basa totalmente en el manejo de eventos entre publicadores y suscriptores.

Modern Technology Stack

Conclusiones

Stack coherente

El stack **Vue.js + FastAPI + MySQL + MQTT** es **coherente y complementario**

Fortalezas individuales

Vue.js maximiza **UX/DX**, **FastAPI** aporta **rendimiento y tipado**, **MySQL** ofrece **persistencia robusta**, **MQTT** habilita **pub/sub eficiente**

Viabilidad práctica

Se ajusta a buenas prácticas y es **viable** para proyectos con **tiempo real, APIs** y IoT

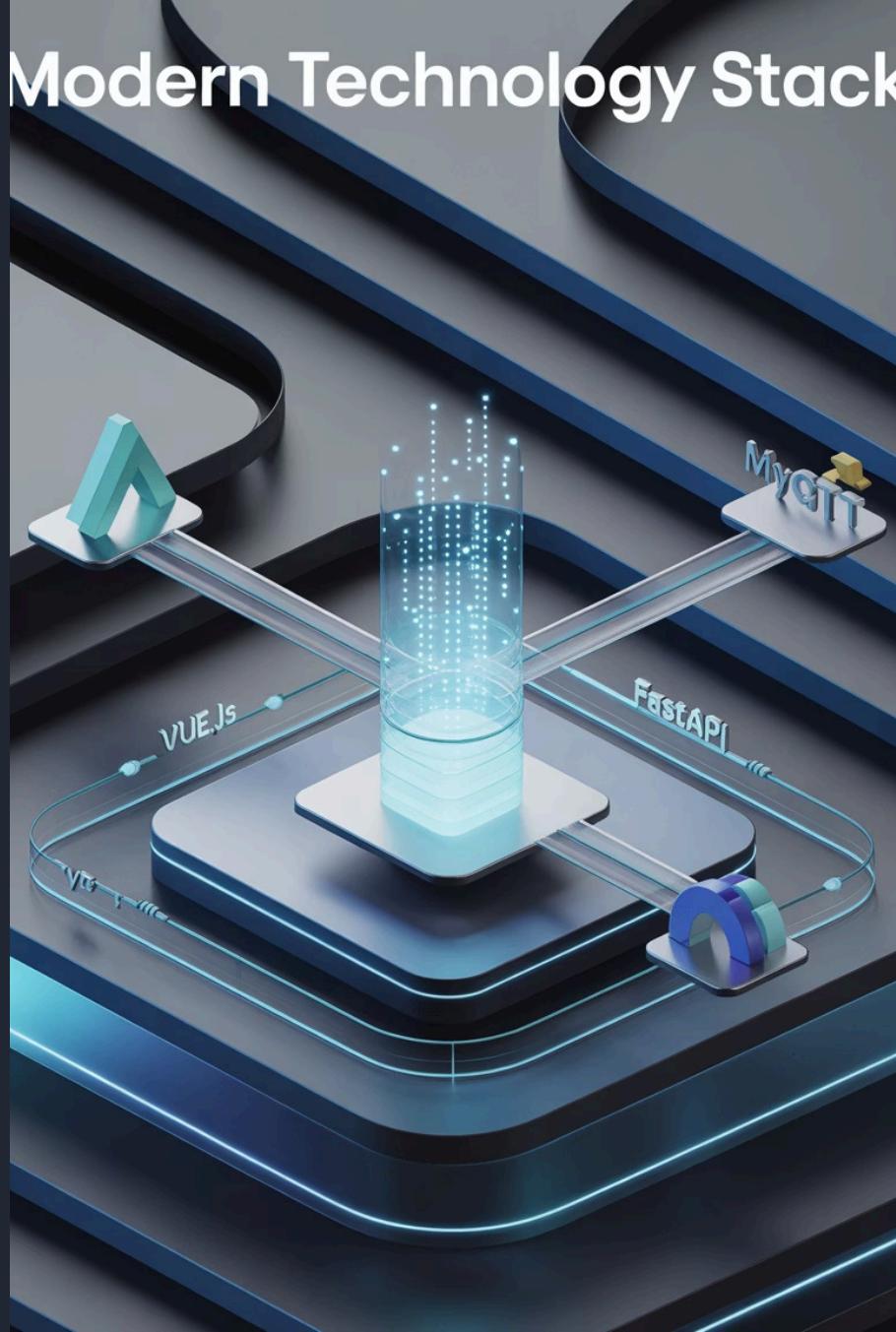


Diagrama Alto Nivel

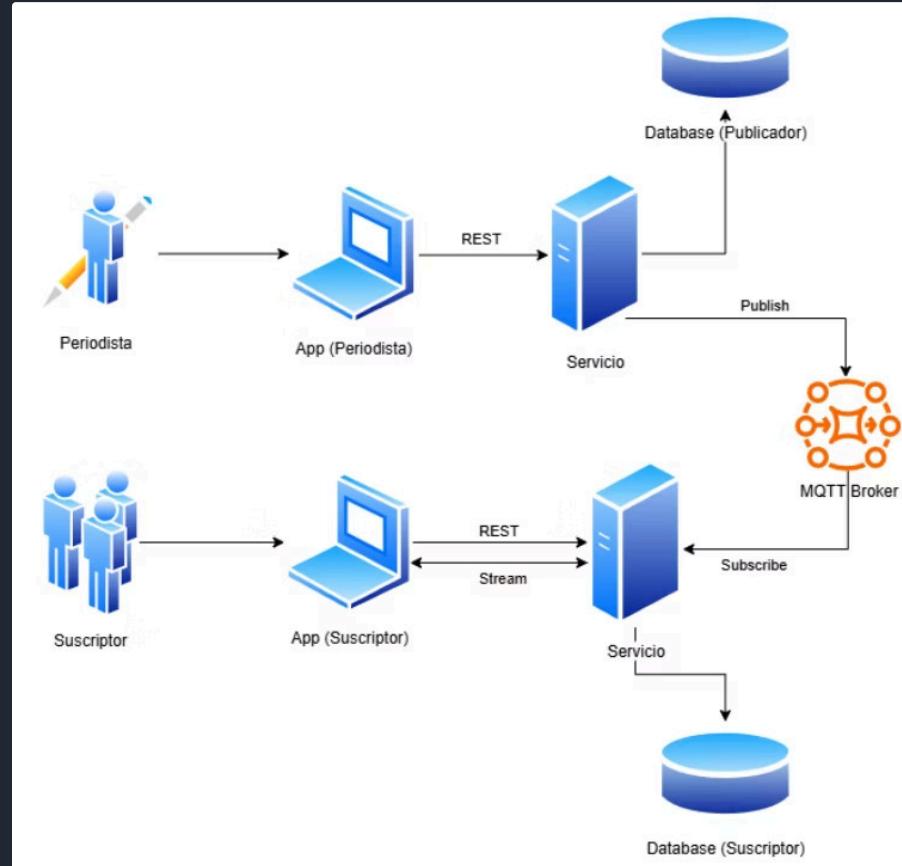


Diagrama de Contexto - C4

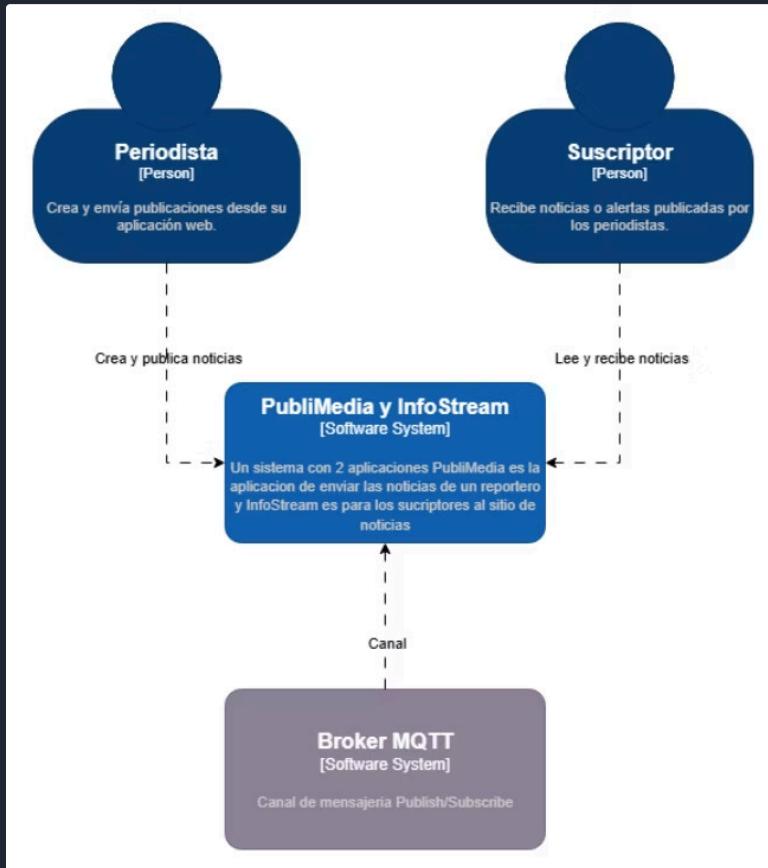


Diagrama de Contenedores - C4

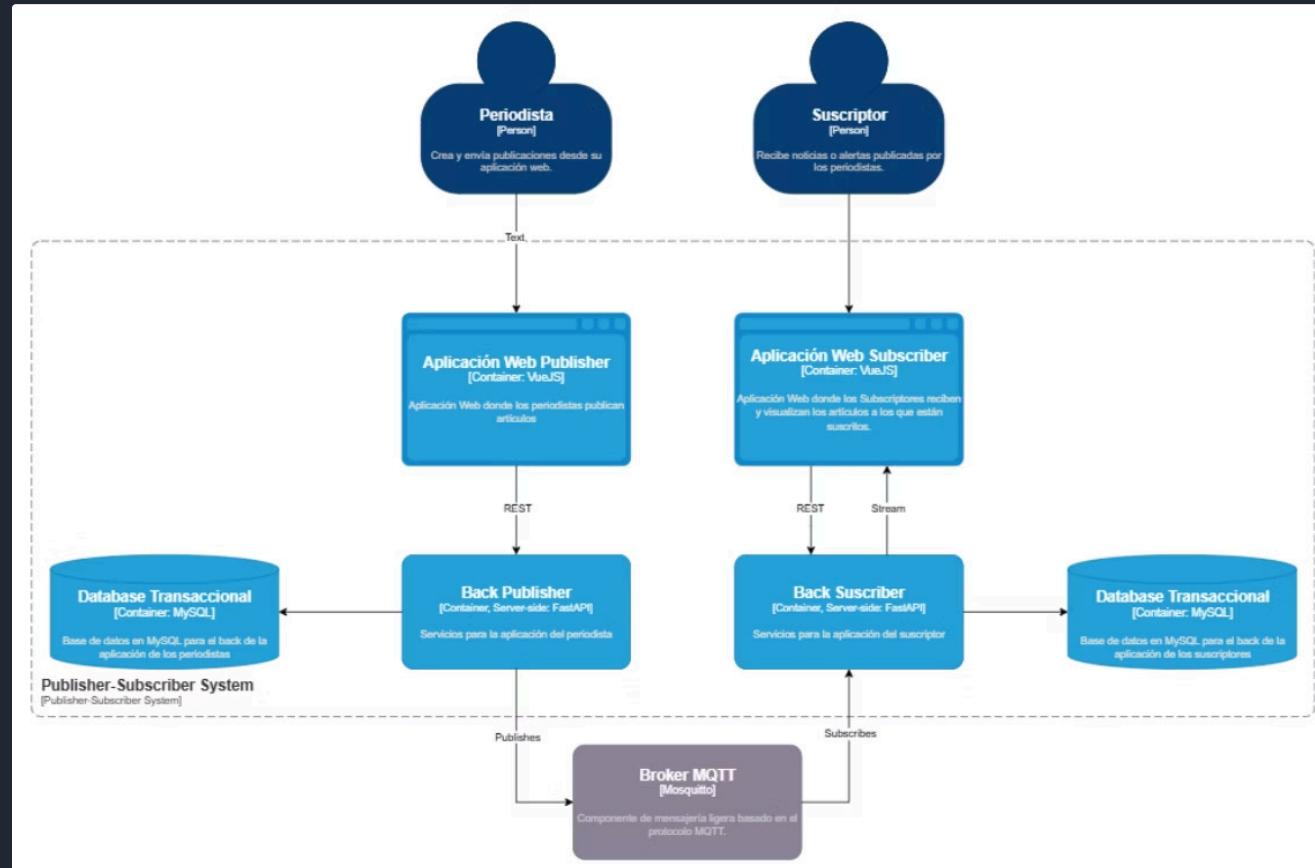


Diagrama de Componentes - C4

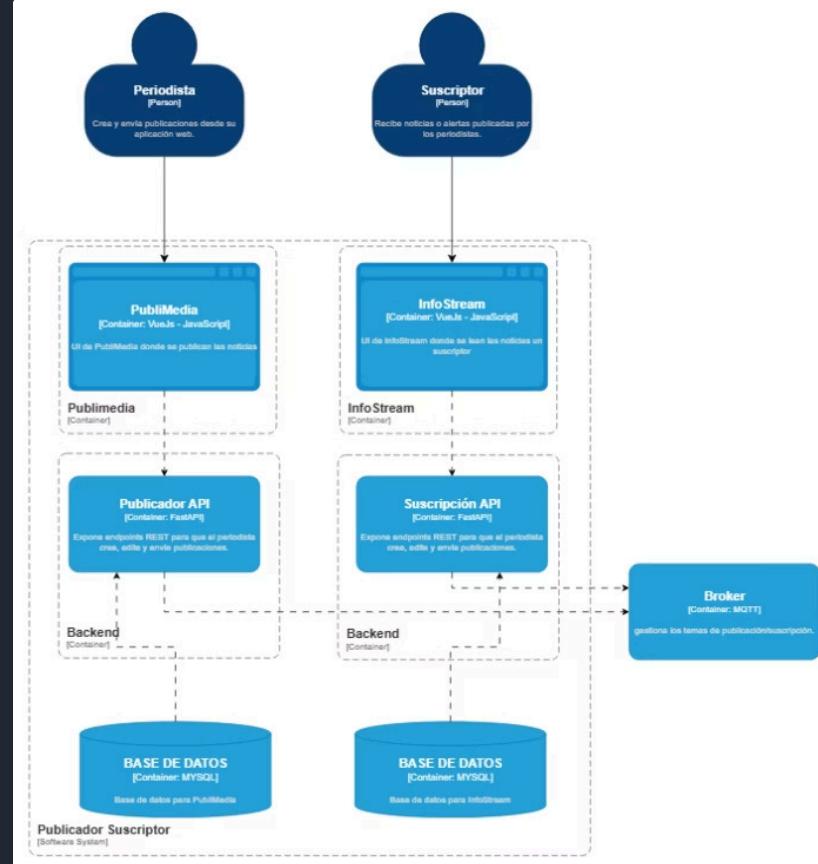


Diagrama de Código Publicador - C4

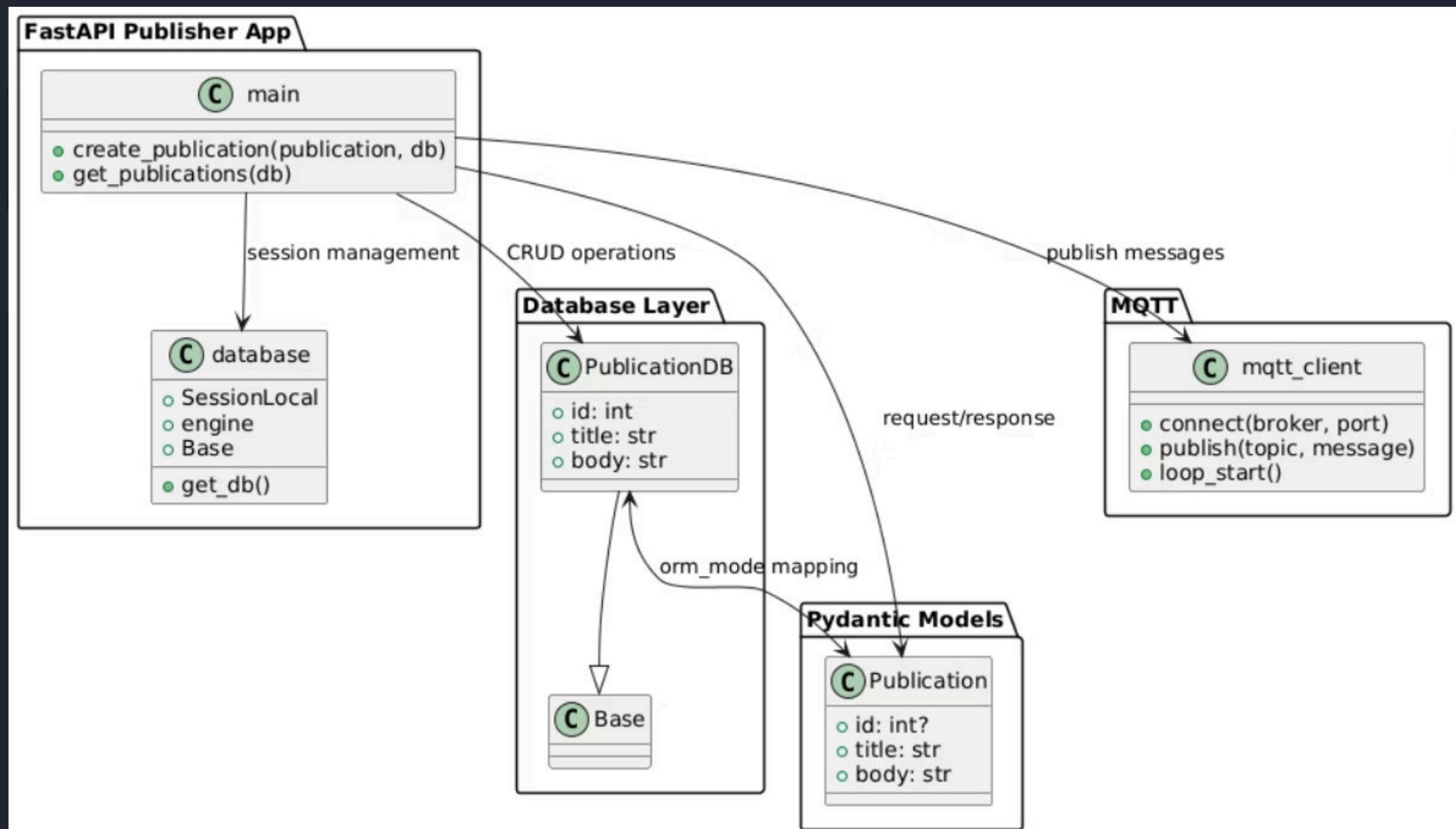


Diagrama de Código Suscriptor - C4

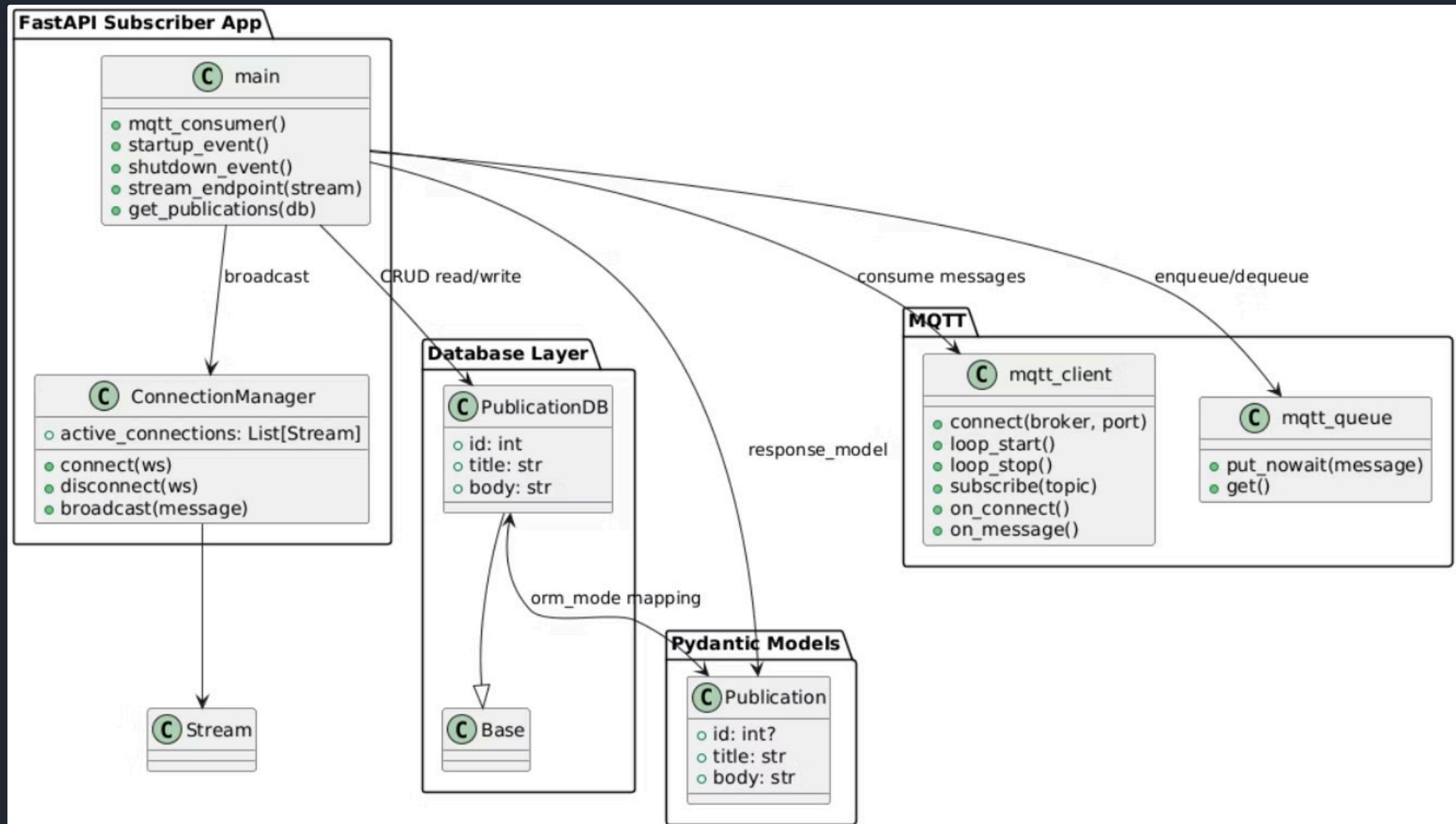


Diagrama Dynamic - C4

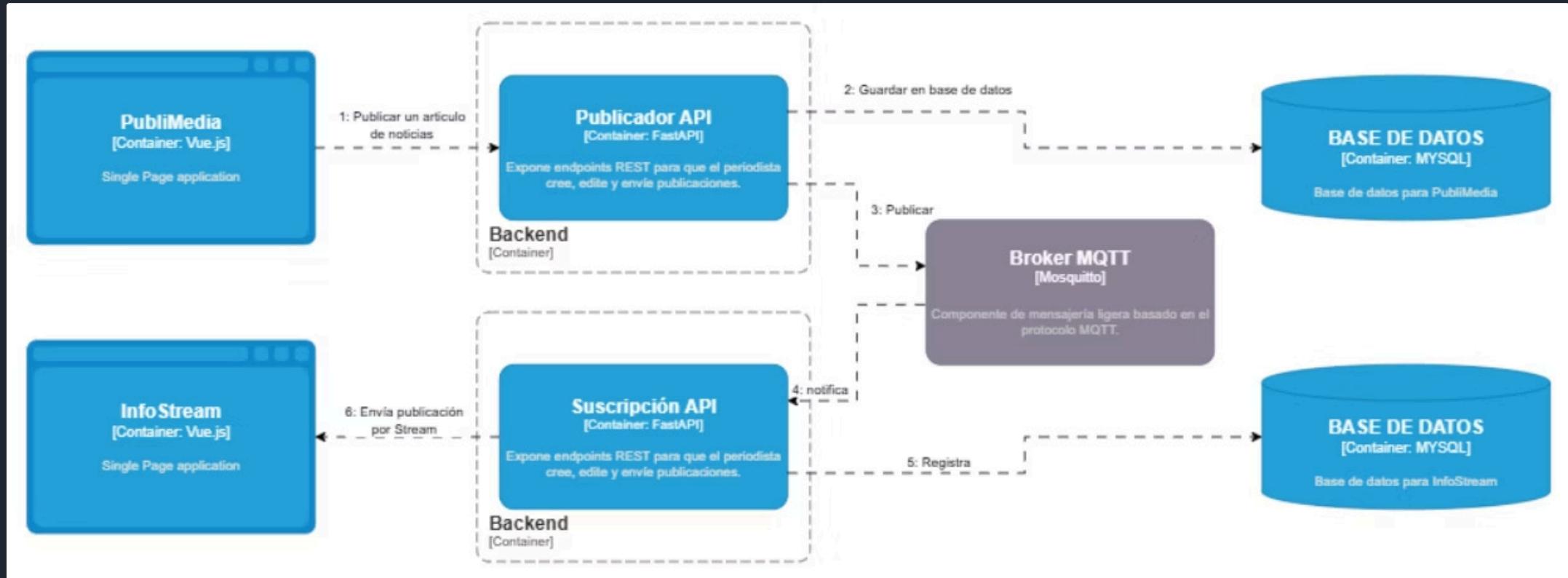


Diagrama Despliegue C4

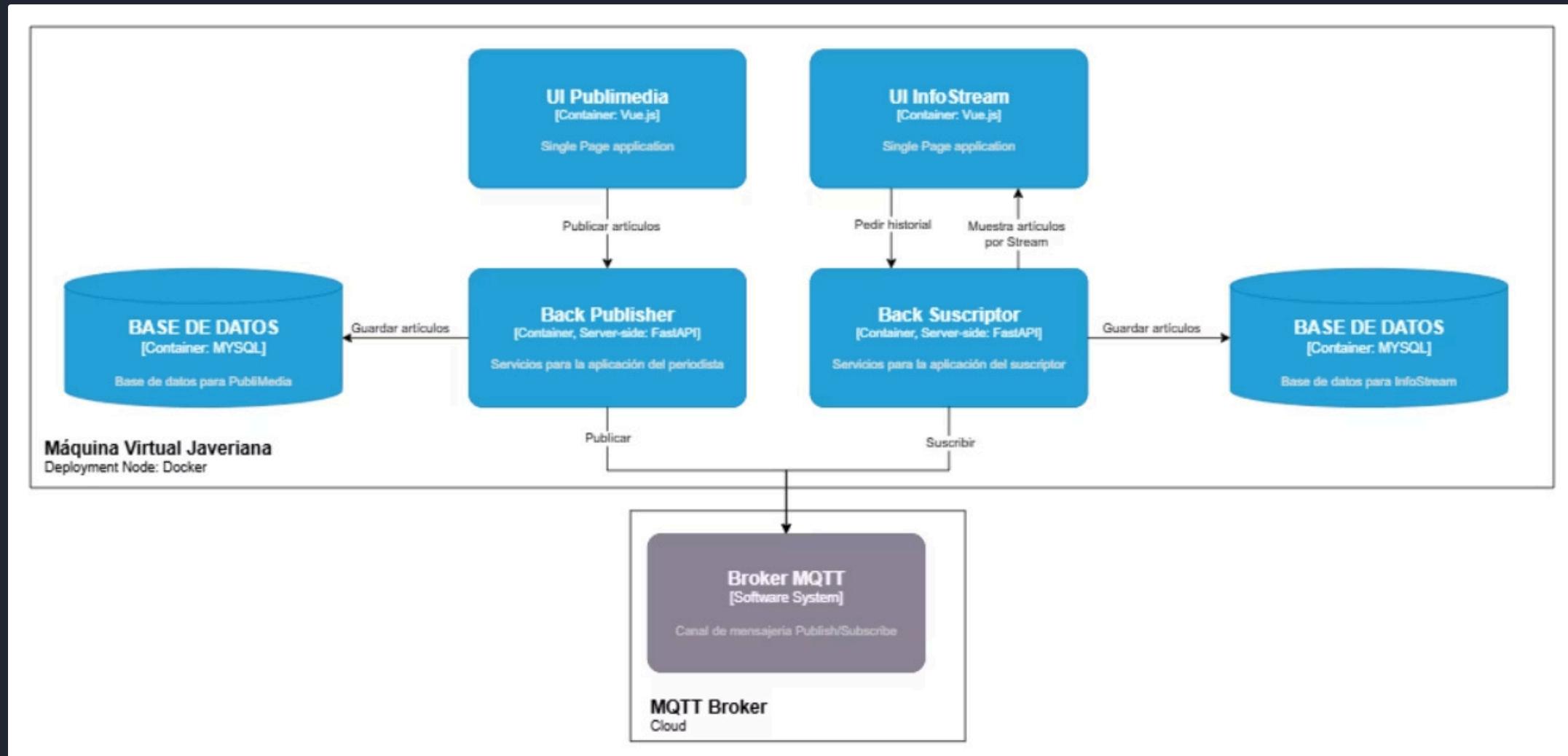


Diagrama de paquetes UML - Publicador

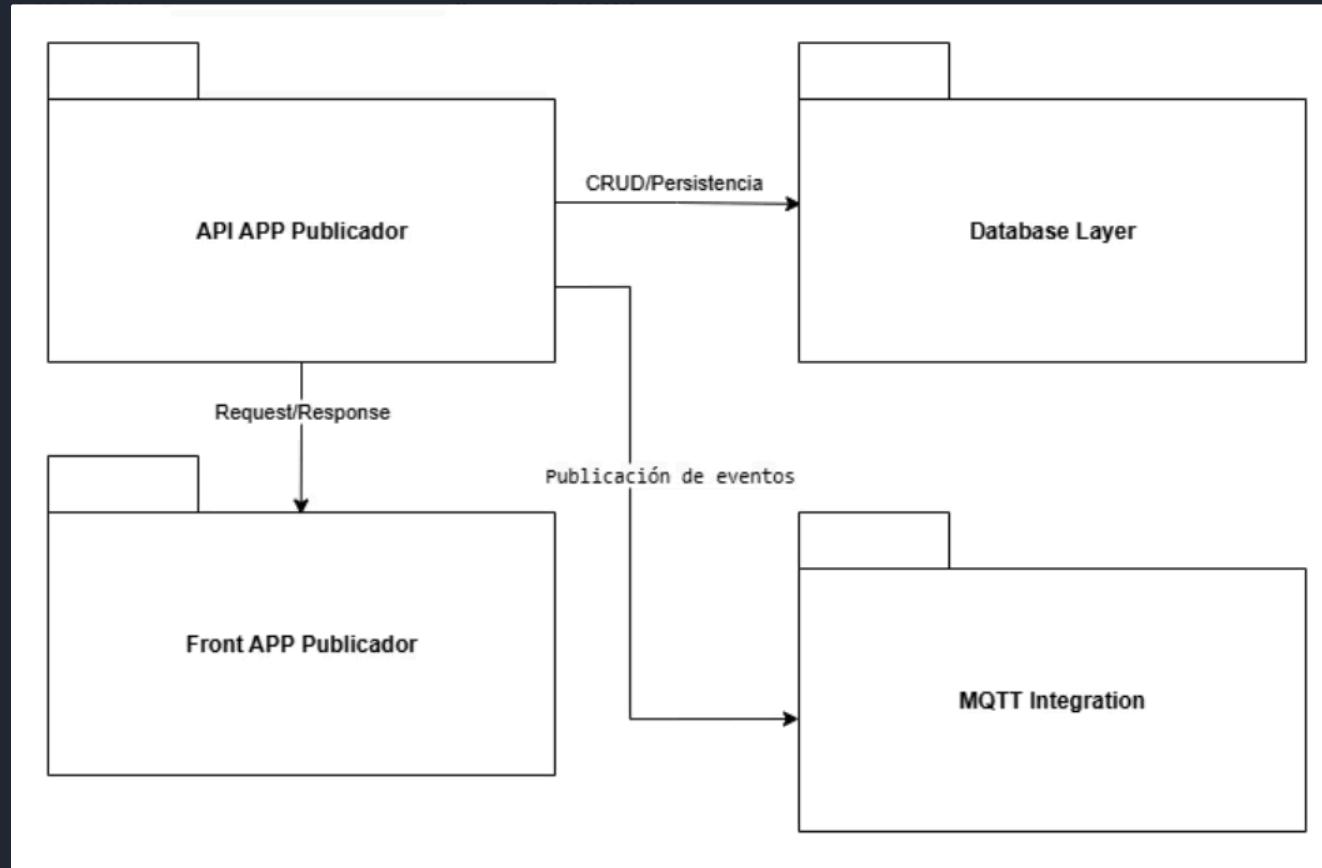
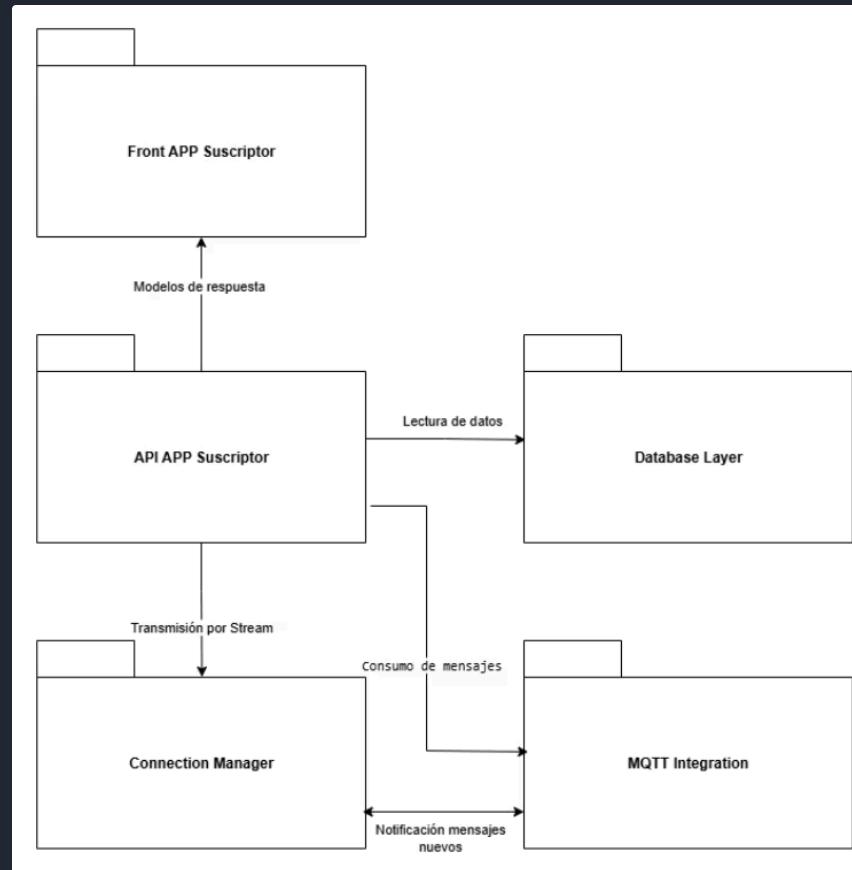


Diagrama de paquetes UML - Suscriptor





Demo