

{dev/talles}

MICROSERVICIOS

UNA COMPARATIVA NEUTRAL

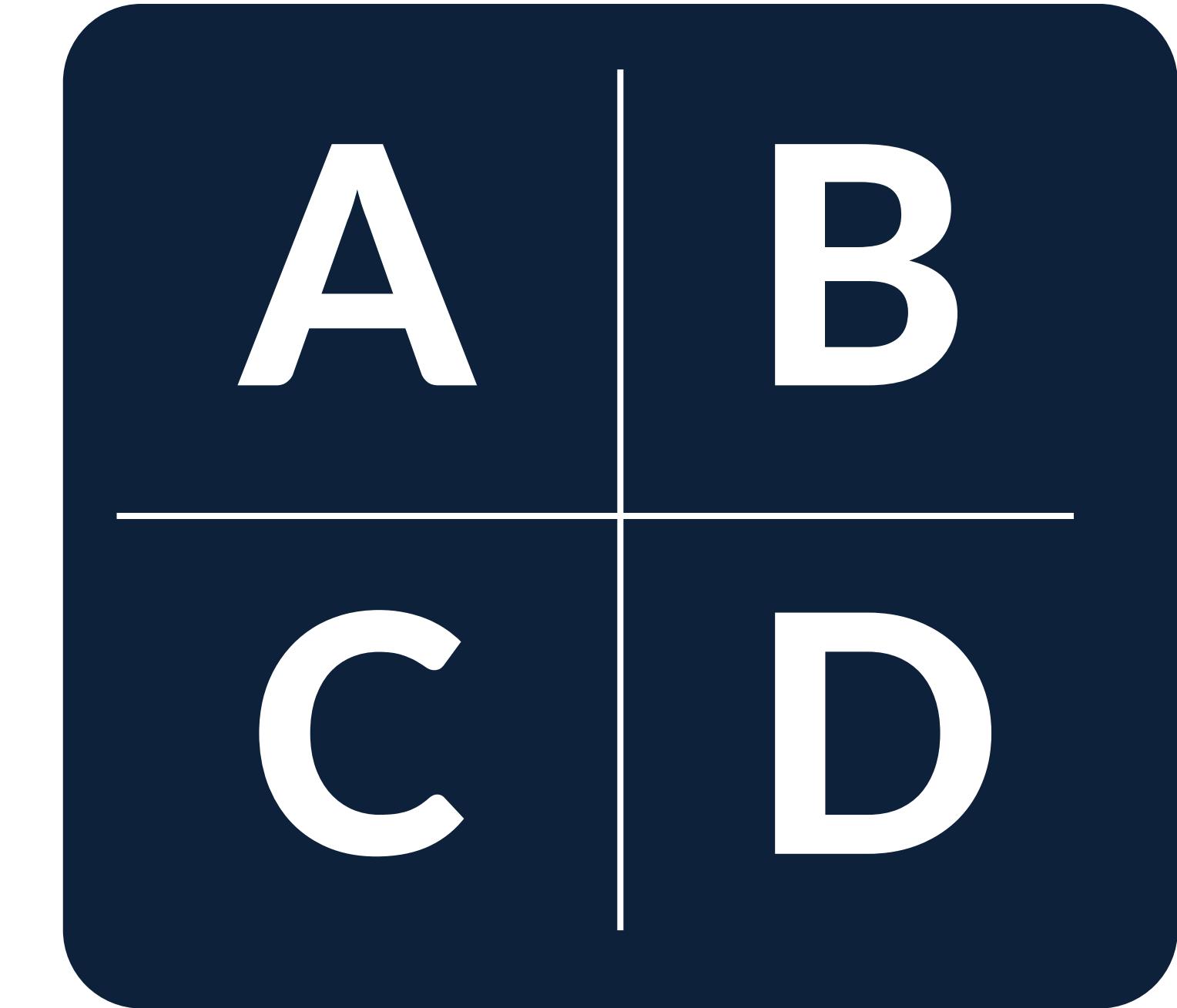


TABLA DE CONTENIDO

- | | |
|--------------------|----------------------|
| 01. Monolítico | 06. Problemas |
| 02. Problemas | 07. Gateways |
| 03. Beneficios | 08. Transportadores |
| 04. Microservicios | 09. Buenas prácticas |
| 05. Beneficios | |



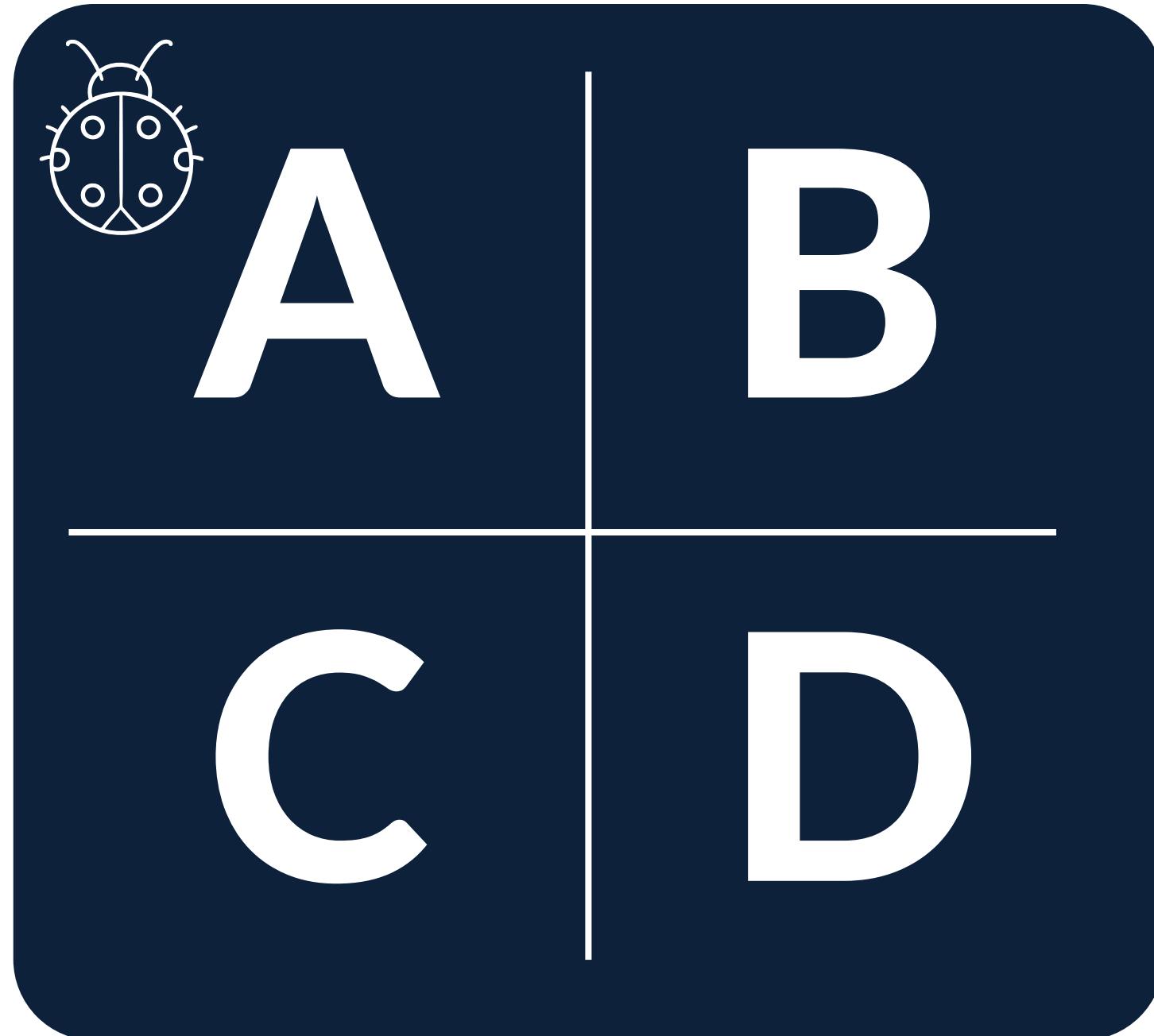
**MONOLITICO
TRADICIONAL**



Todo el código está interconectado entre sí

CONS

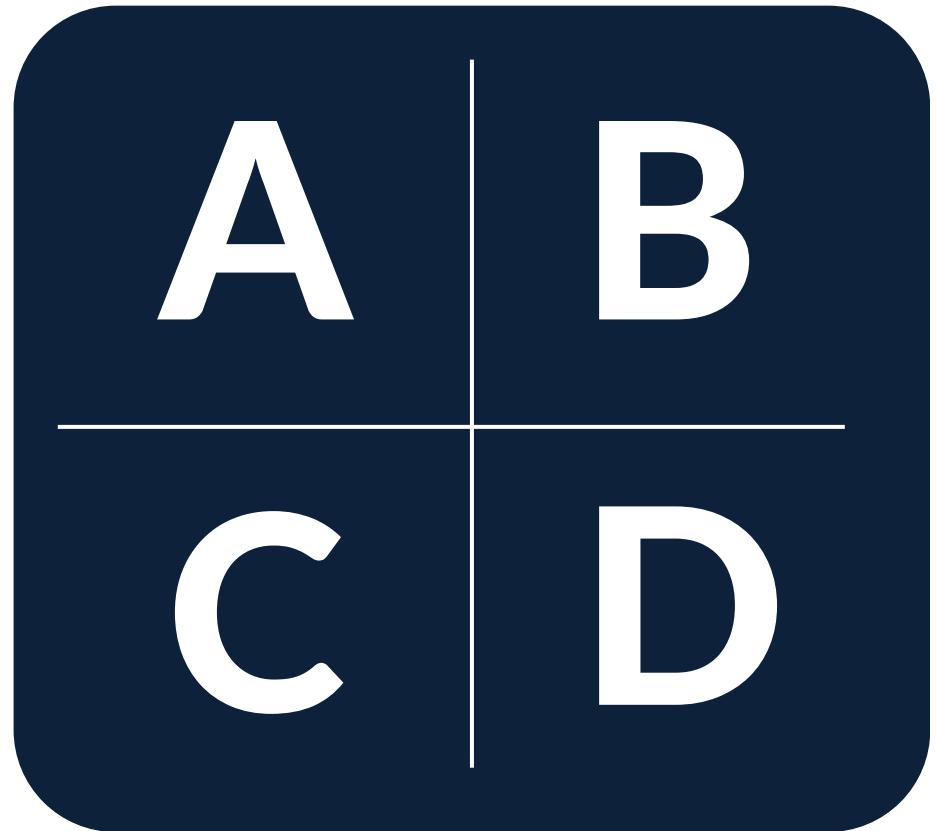
**MONOLITICO
TRADICIONAL**



Ante un error en el paquete A tenemos que redesplegar toda la app

CONS

PUNTOS A CONSIDERAR



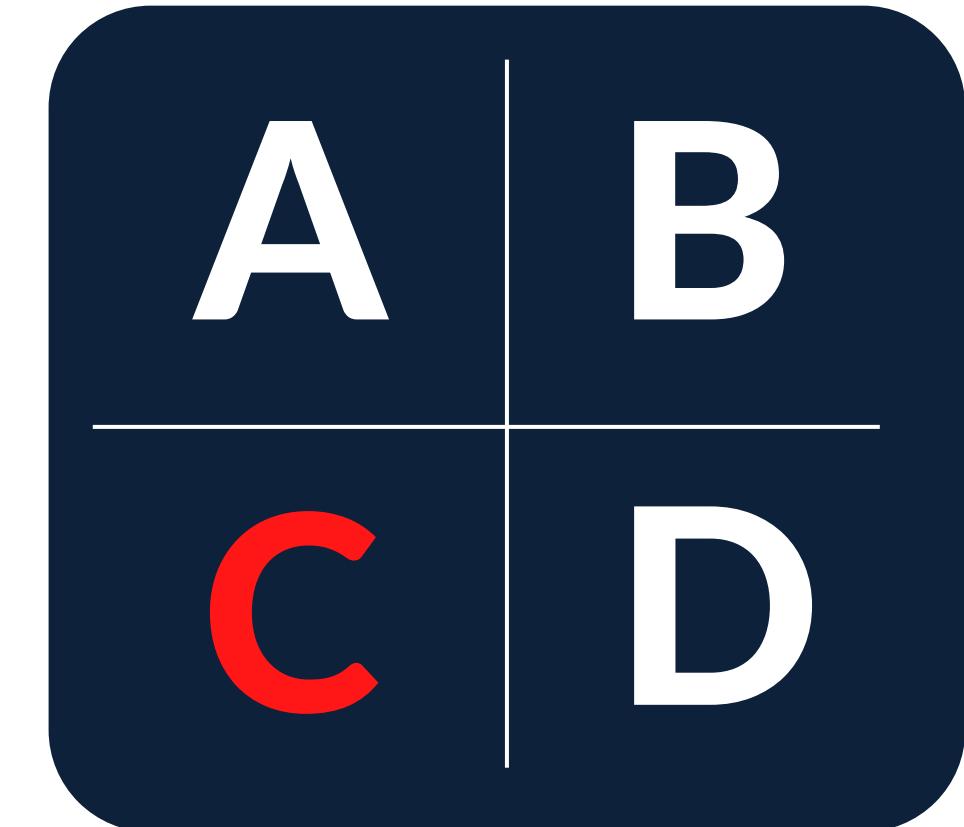
MONOLITICO

CONS

PUNTOS A CONSIDERAR

MONOLITICO

Escalamiento Vertical: Más RAM, Disco Duro más rápido... ESCALAMIENTO VERTICAL



Mucho estrés

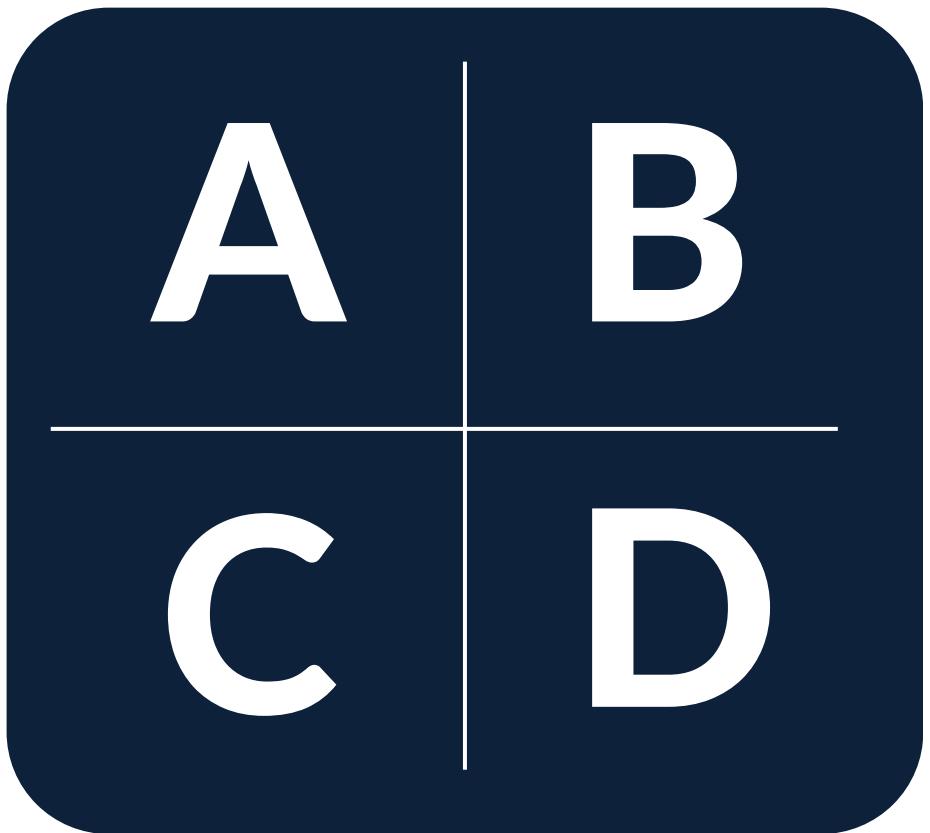
No podemos escalar C

Sin escalar todo el monolítico

Se entorpece A, B y D

CONS

PUNTOS A CONSIDERAR



D v2.0

Nueva versión



MONOLITICO

CONS

PUNTOS A CONSIDERAR



No se puede actualizar D

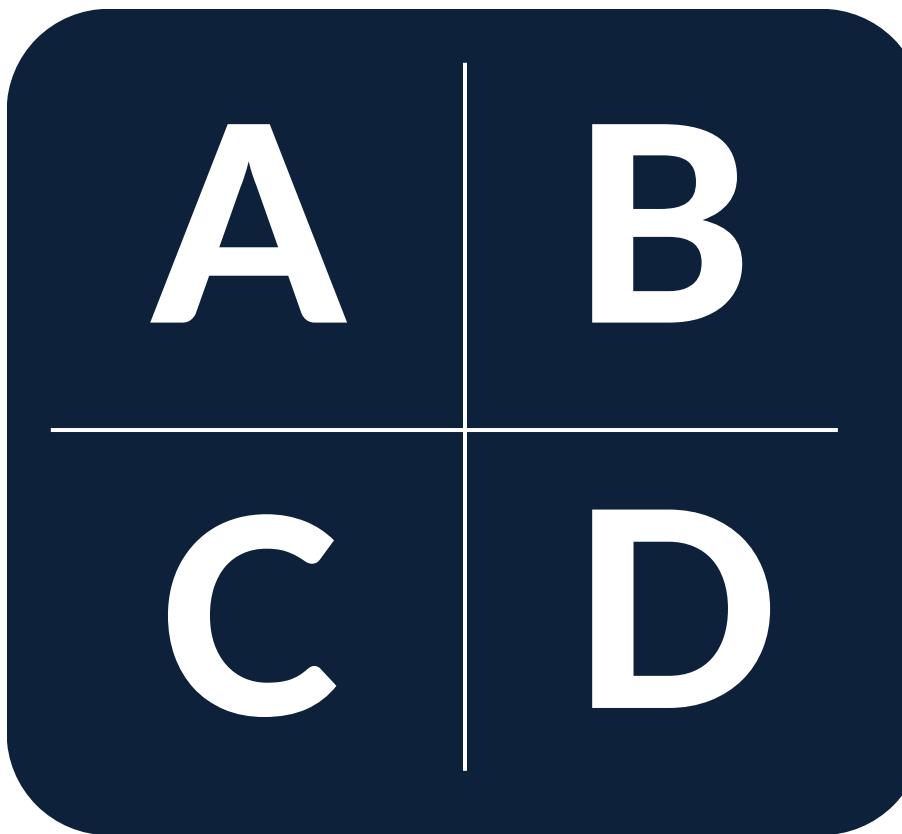
Sin redesplegar A,B y C también

Si tenemos que revertir cambios, también revertimos A, B y C

MONOLITICO

CONS

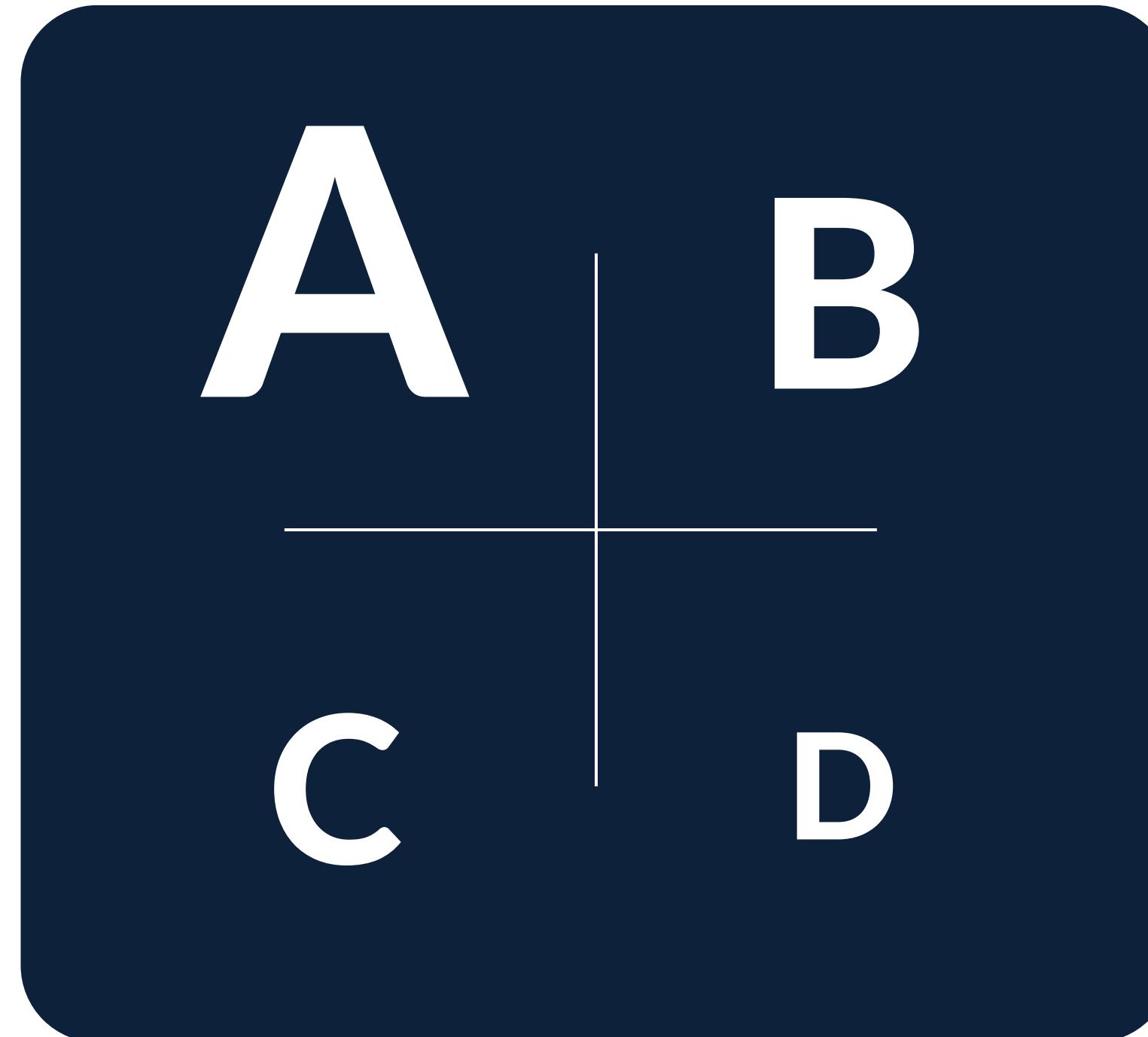
PUNTOS A CONSIDERAR



MONOLITICO

CONS

PUNTOS A CONSIDERAR



- Código base va creciendo.
- Difícil de ejecutar localmente.

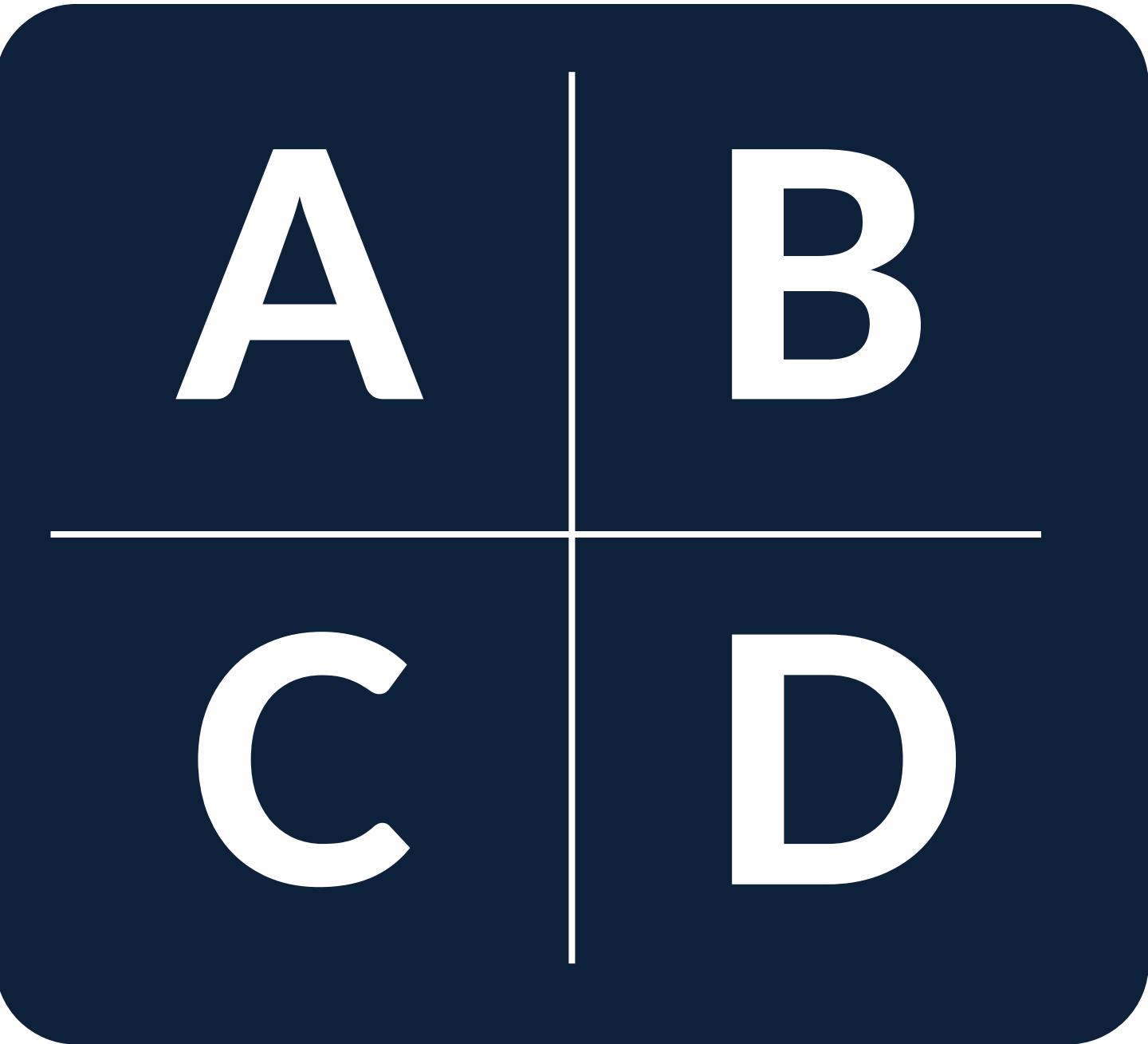
Va creciendo de forma disparate.

Puede que haya partes con información sensible que no queremos que vea el desarrollador y la va a ver.

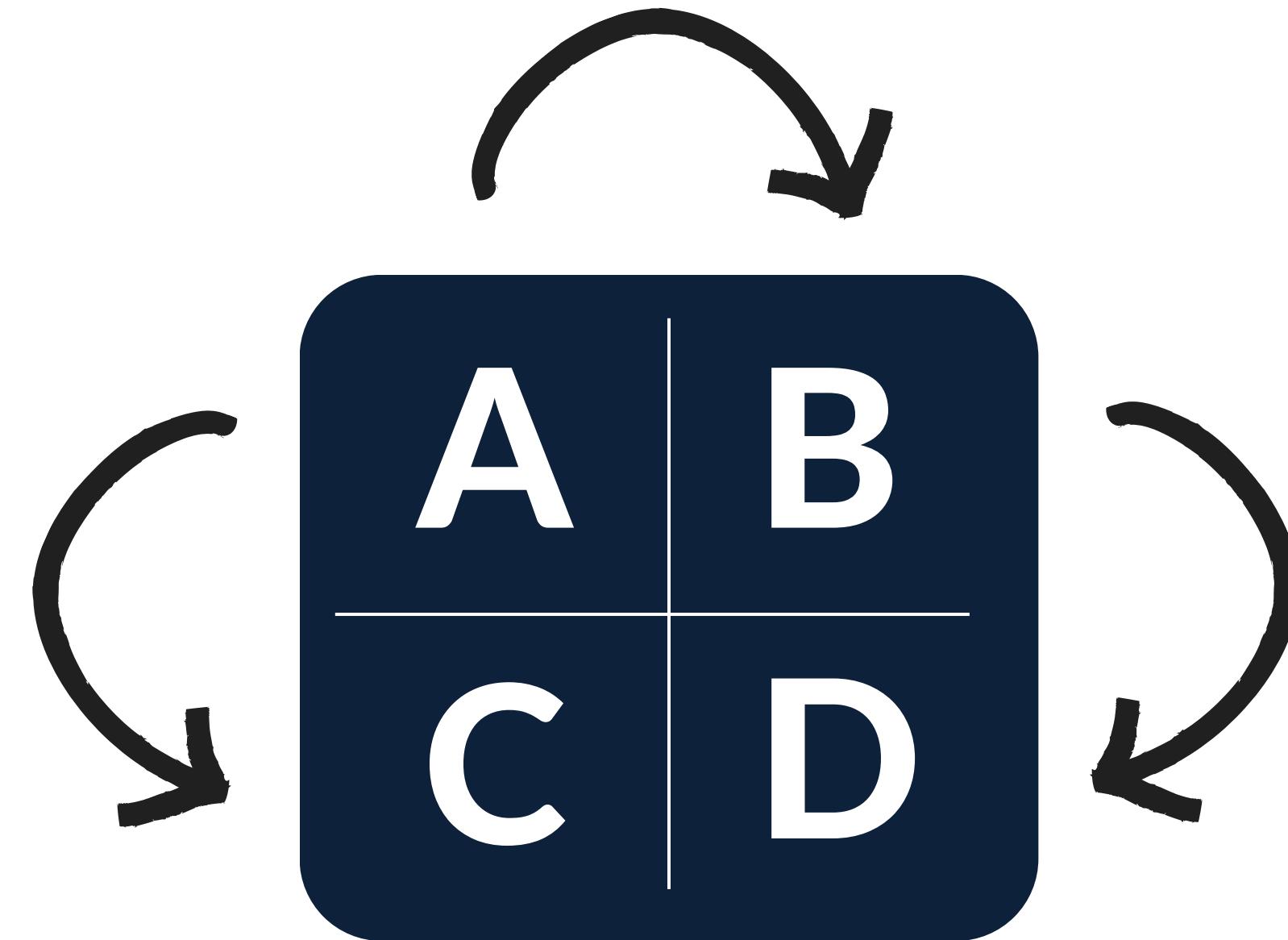
MONOLITICO

PROS

**MONOLITICO
TRADICIONAL**



PROS



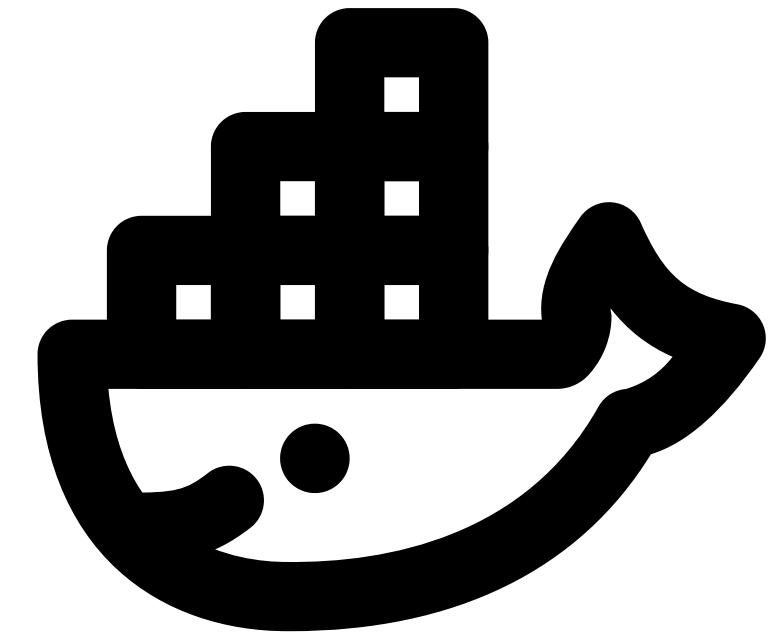
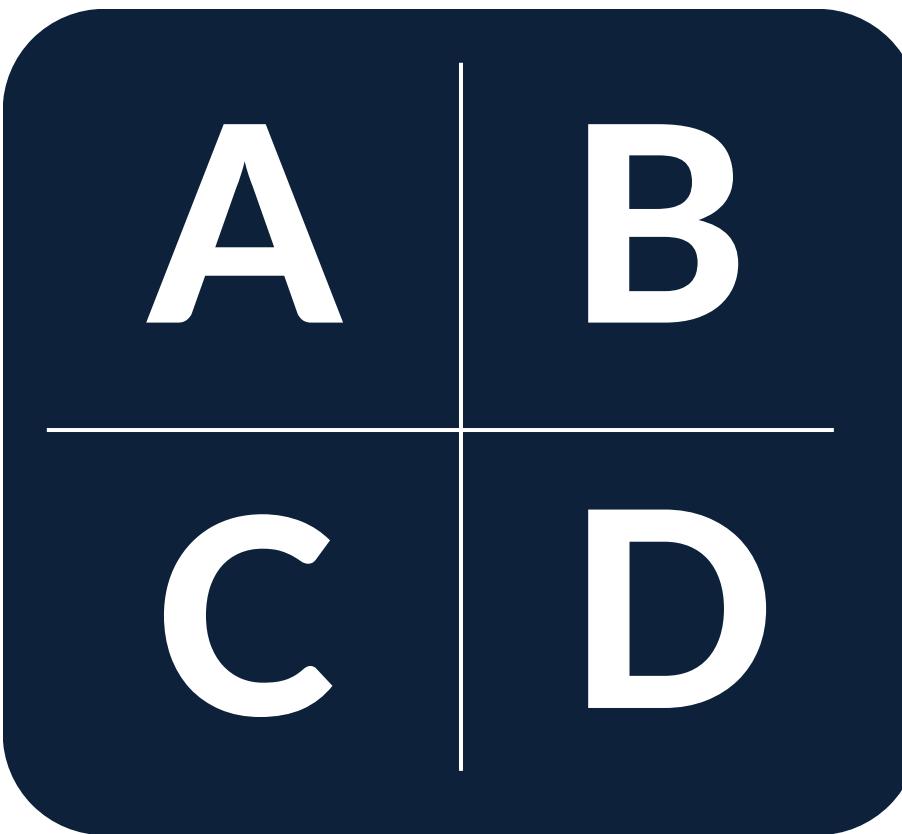
Cero latencia

COMUNICACIÓN INSTANTÁNEA ENTRE LAS PARTES

No comunicación HTTP, ni FTP, ni TCP, ni conexión con mensajería...

MONOLITICO

PROS

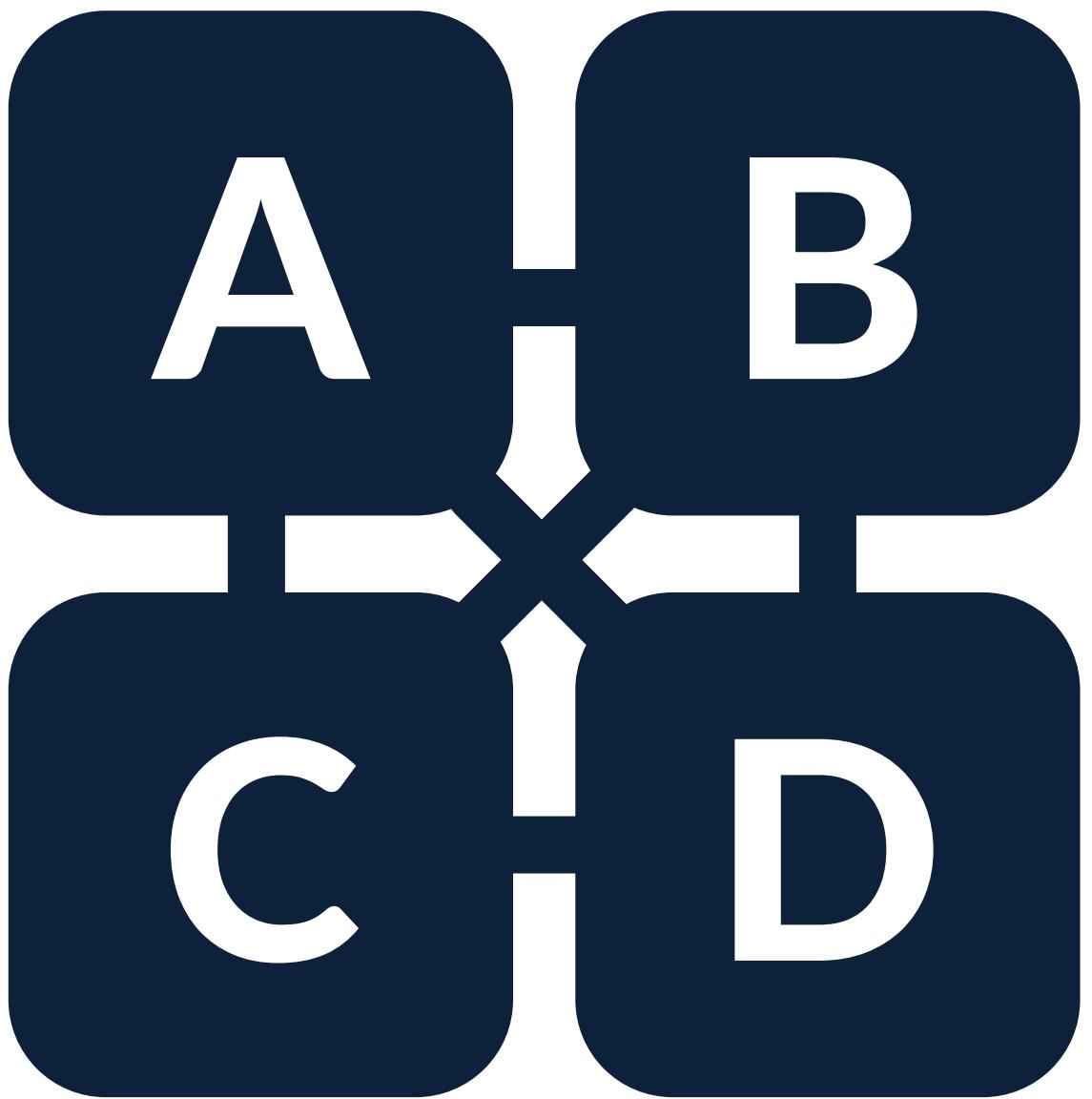


MONOLITICO

Montar todo en una máquina o contenedor

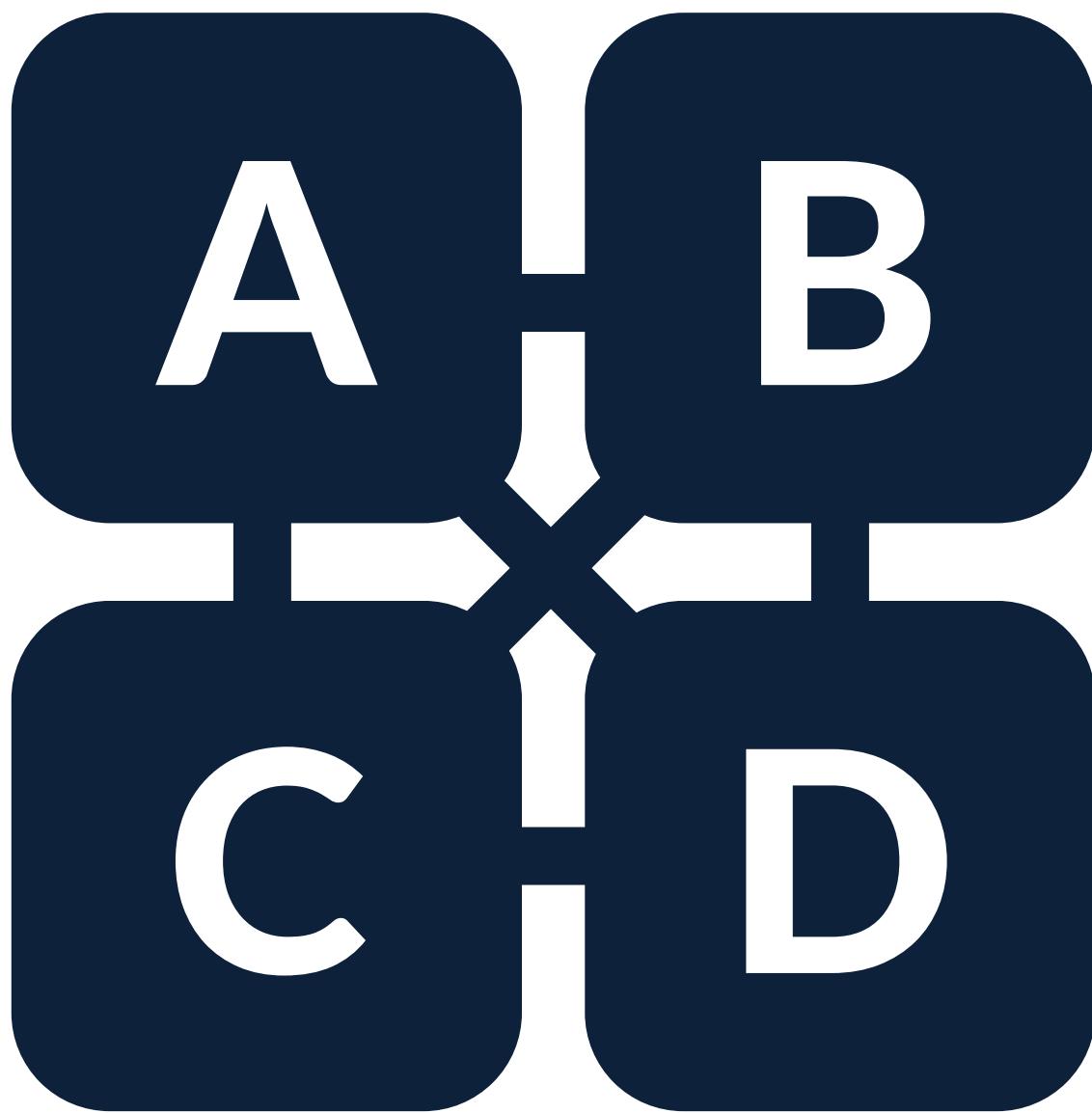
MICROSERVICIOS

PROS Y CONS



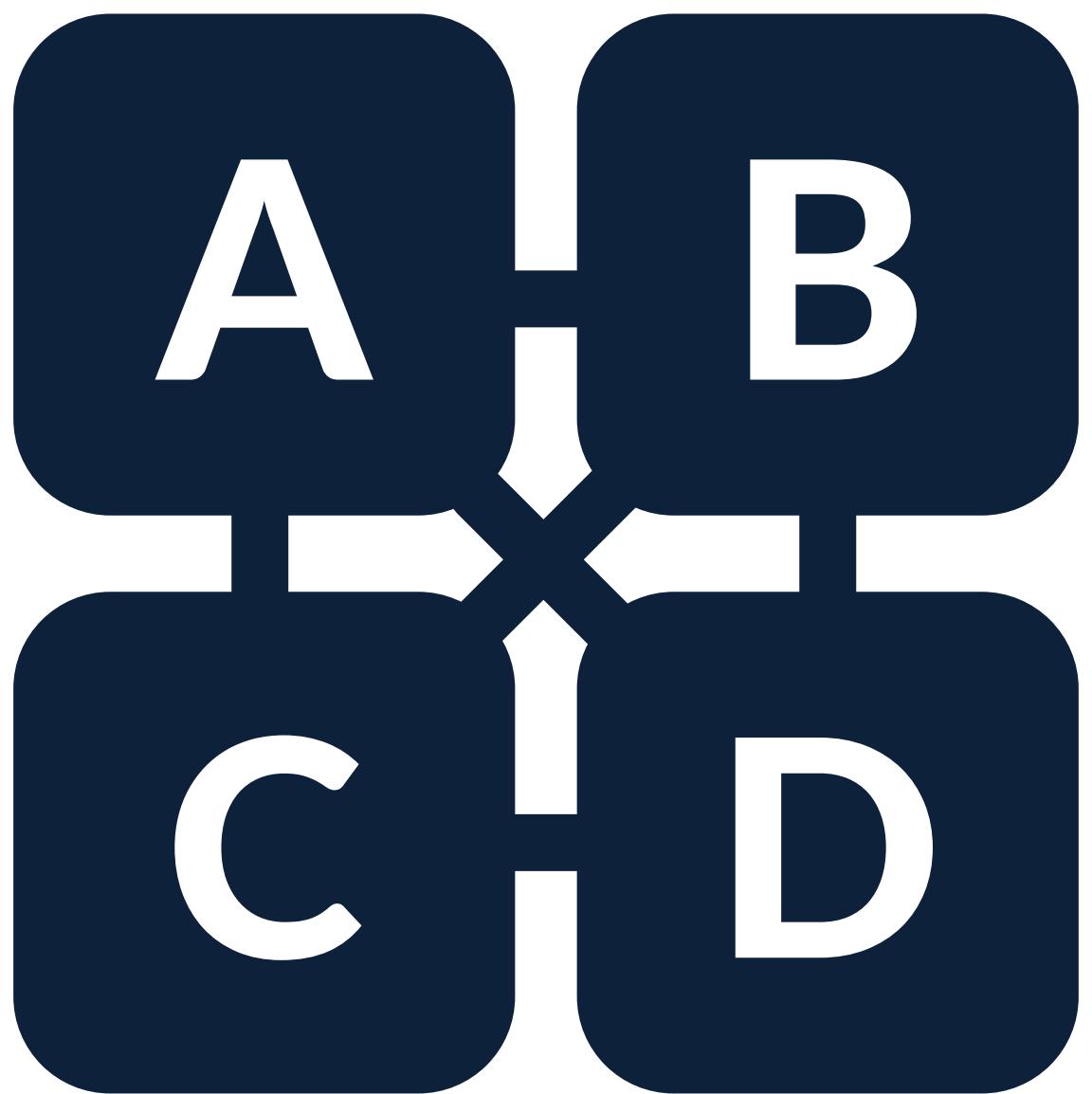
MICROSERVICIOS

NO ES UNA BALA DE PLATA



**Los microservicios resuelven algunos problemas
pero incorporan otros**

PROS





PROS



TODOS LOS MICROSERVICIOS SON INDEPENDIENTES, ESCALABLES Y DISTRIBUÍDOS

Crecen de forma independiente



PROS



Si, por ejemplo, A está presentando problemas, deberíamos poder resolverlos sin afectar B ni C ni D



PROS



Si A está teniendo mucho estrés, podríamos realizar un escalamiento horizontal, es decir, agregar réplicas del microservicio A y un balanceador de carga, para atender el volumen de carga que nuestros usuarios están demandando



PROS



PROS



Los microservicios pueden estar escritos en diferentes lenguajes, y se comunican mediante API Rest, o FTP o TCP o colas o mensajería...





PROS

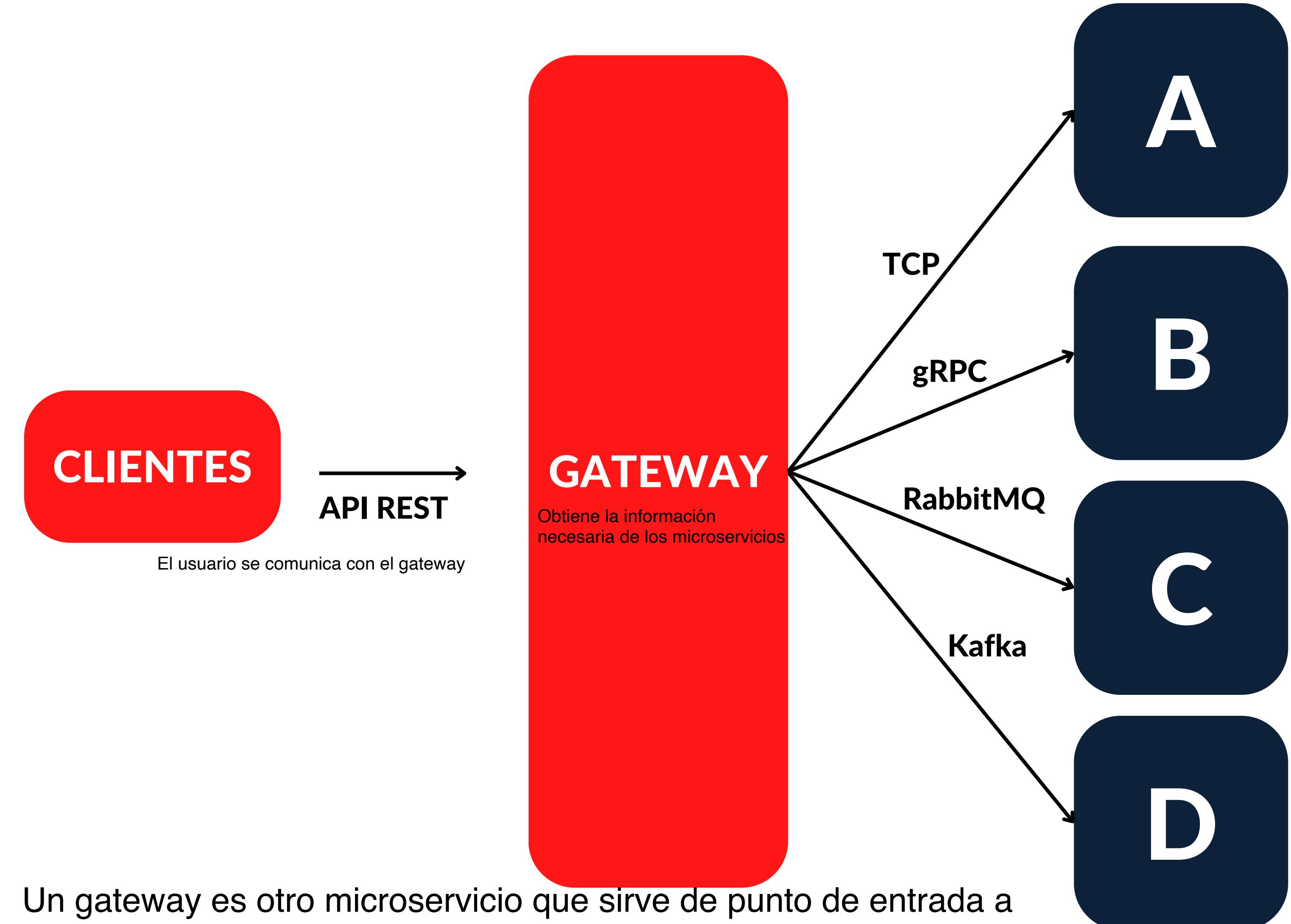


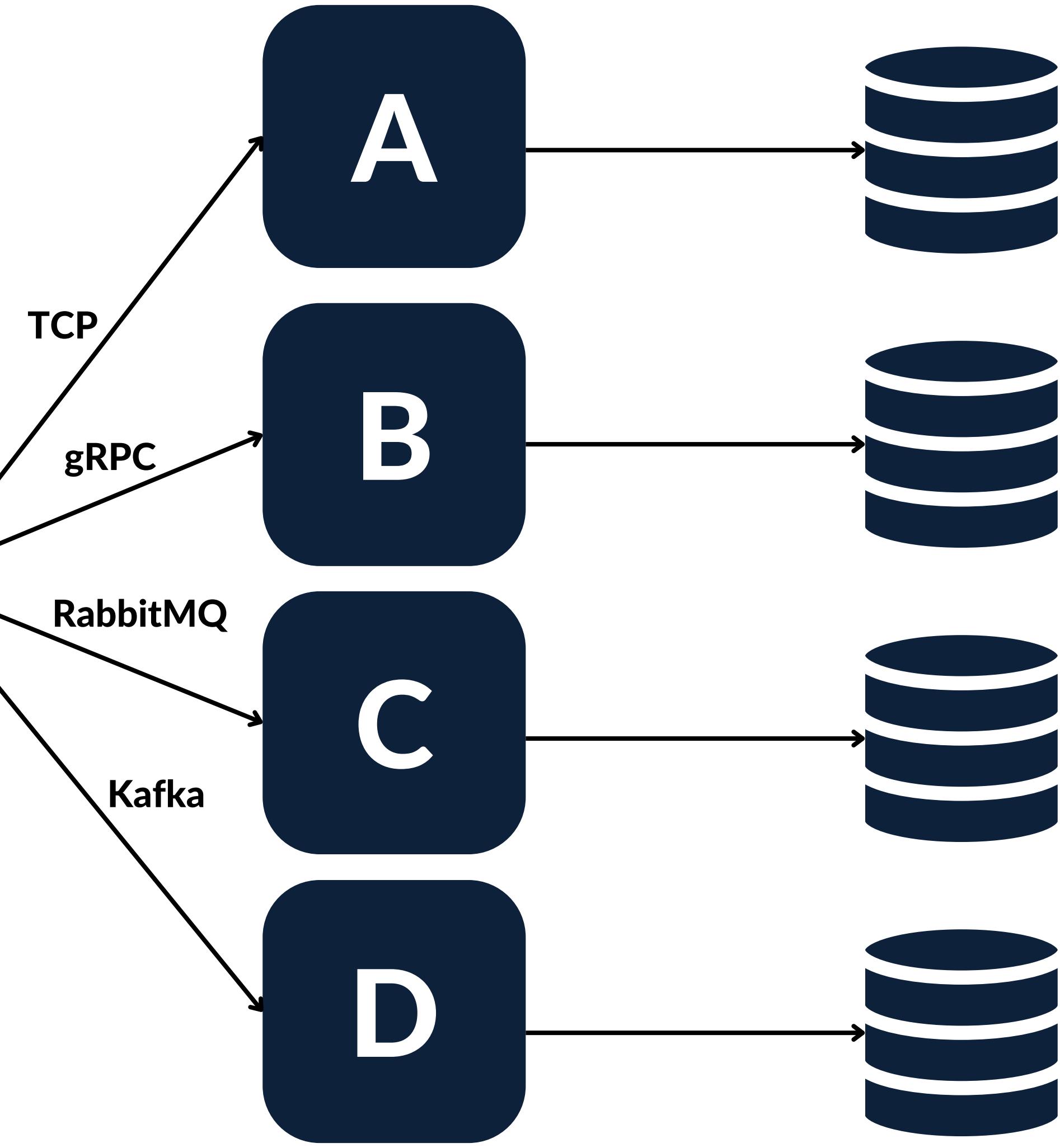
**PUEDEN ESTAR AISLADOS Y COMUNICARSE MEDIANTE REST,
SER HÍBRIDOS O USAR UN CANAL DE COMUNICACIÓN ESPECÍFICO**



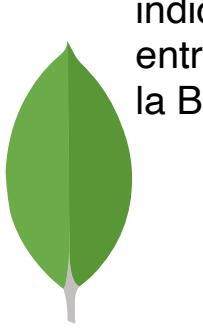
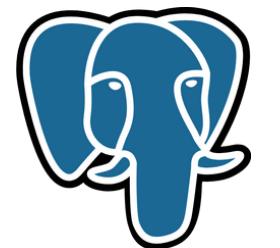
Los microservicios se pueden estructurar de tal manera que sean fáciles de descubrir, y a la vez incluir una barrera que nos sirva para agrupar la comunicación hacia ellos







Microsoft®
SQL Server®



Para que los microservicios puedan escalar y mantener la independencia, las buenas prácticas indican que deben tener BBDD independientes entre sí, es decir, la data de C no se encuentra en la BBDD que tiene A

CONS

A

B

C

D

F

A

B

CONS

A veces nos olvidamos de que existen ciertos microservicios
y volvemos a realizar esa funcionalidad de nuevo

C

J

G

K

H

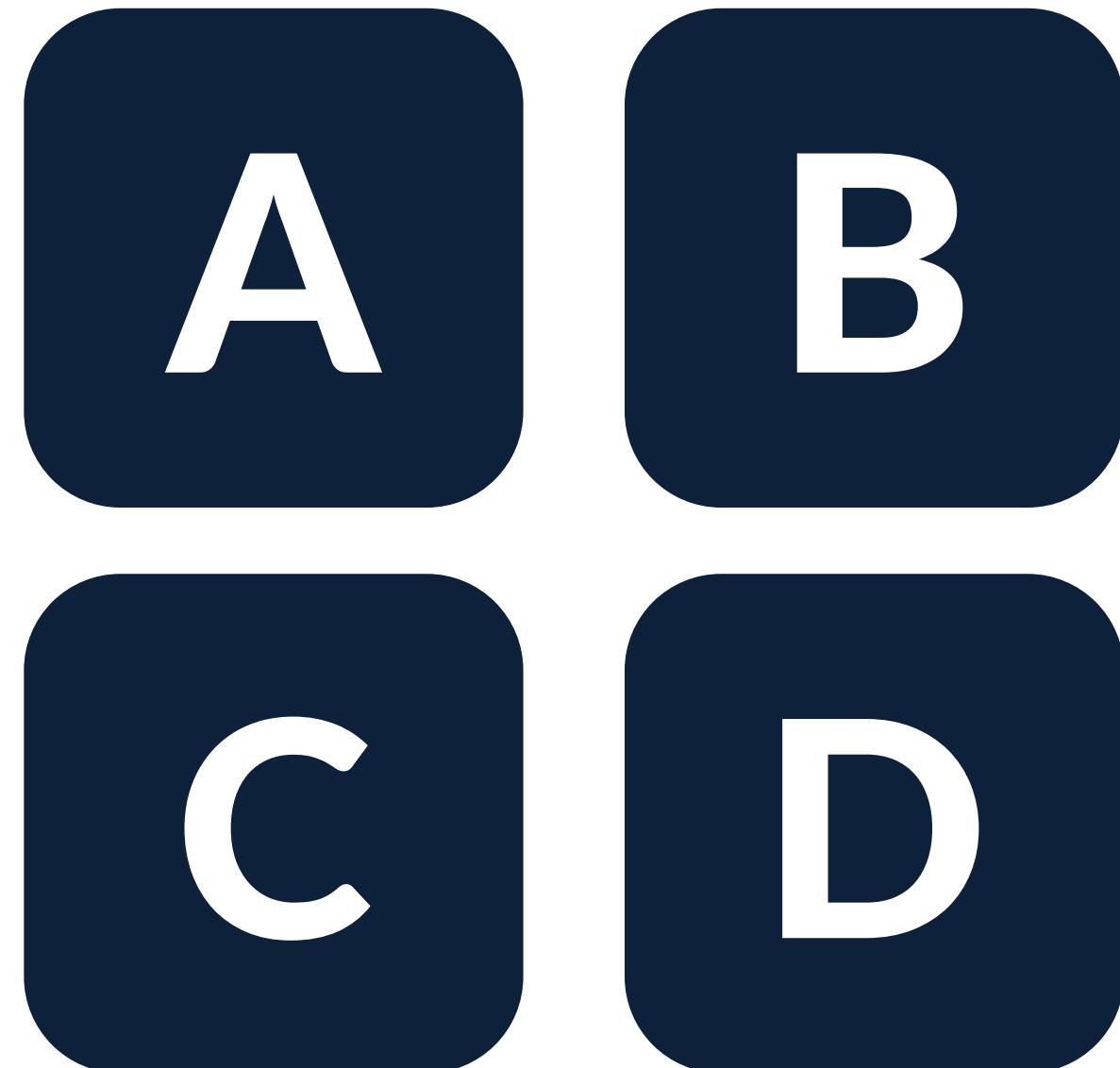
I

D

E

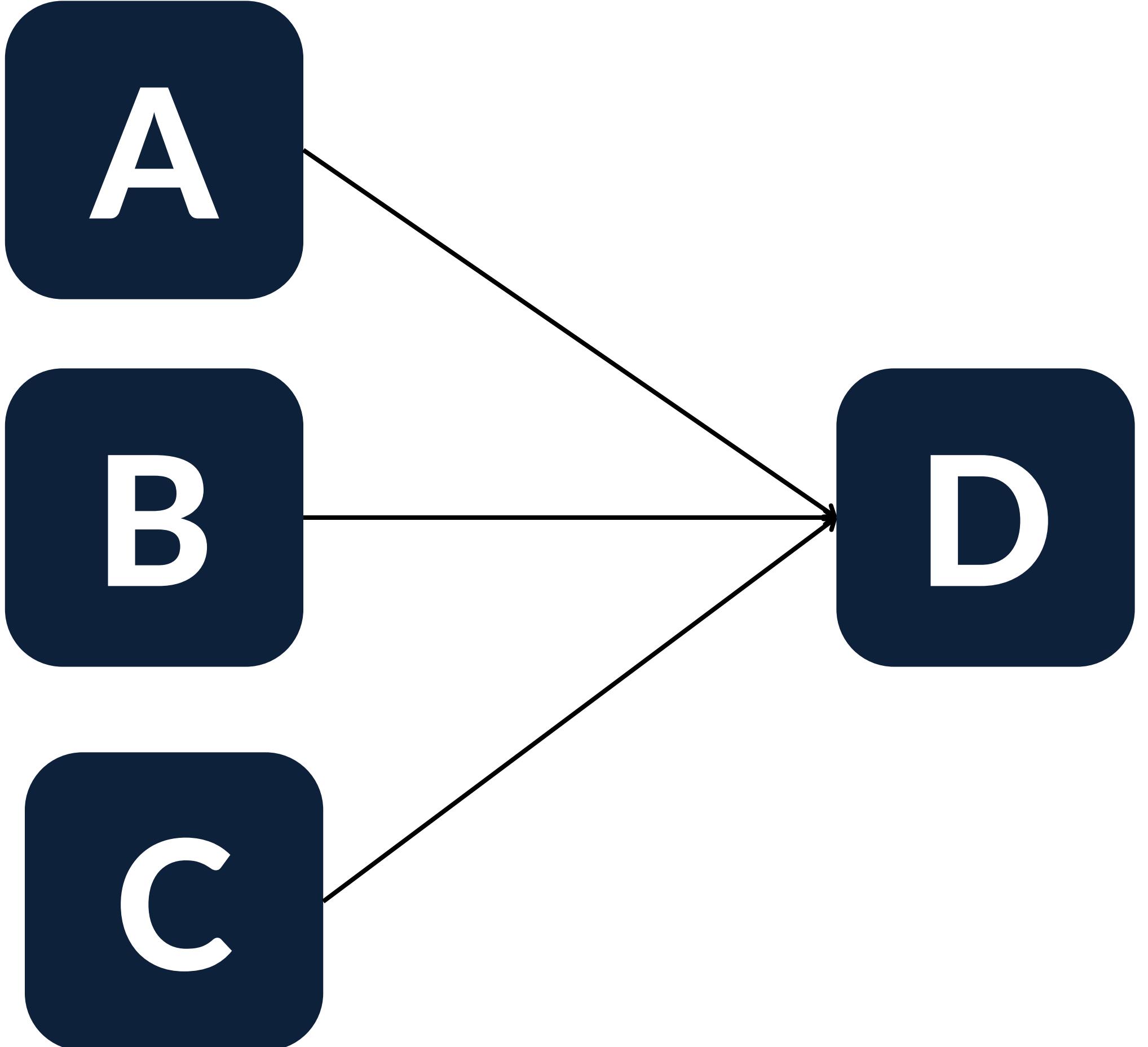
ENCONTRARLOS

CONS



La comunicación interna entre
microservicios incrementa la
latencia

LATENCIA



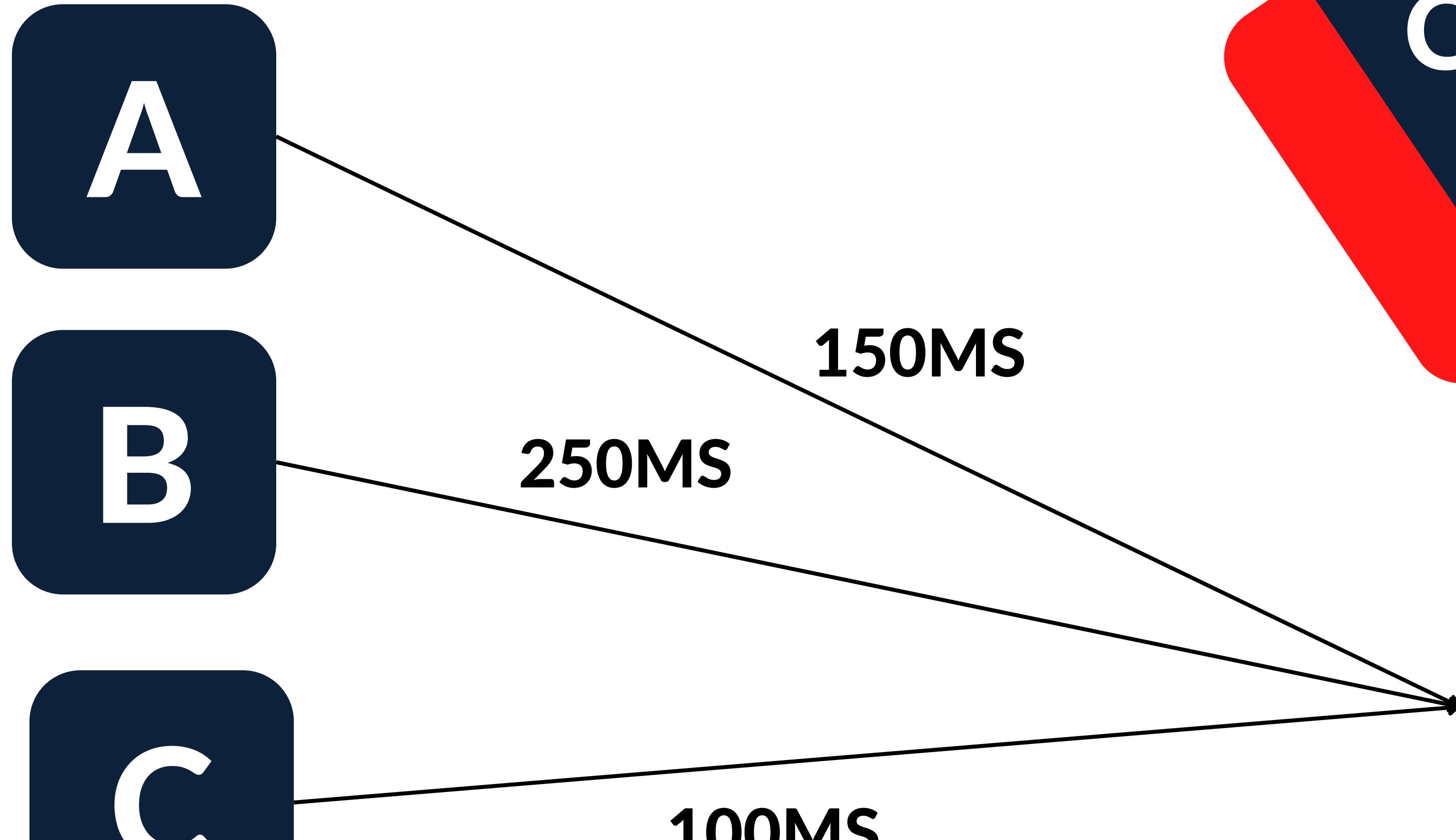
CONS

LATENCIA

A

B

C



LATENCIA

CONS

SI SE REQUIERE ALGO DE LOS SERVICIOS A,B,C



Si se puede hacer todo en paralelo, la latencia corresponde al microservicio que responda más lento. Si no, hay que sumar tiempos



150MS



250MS

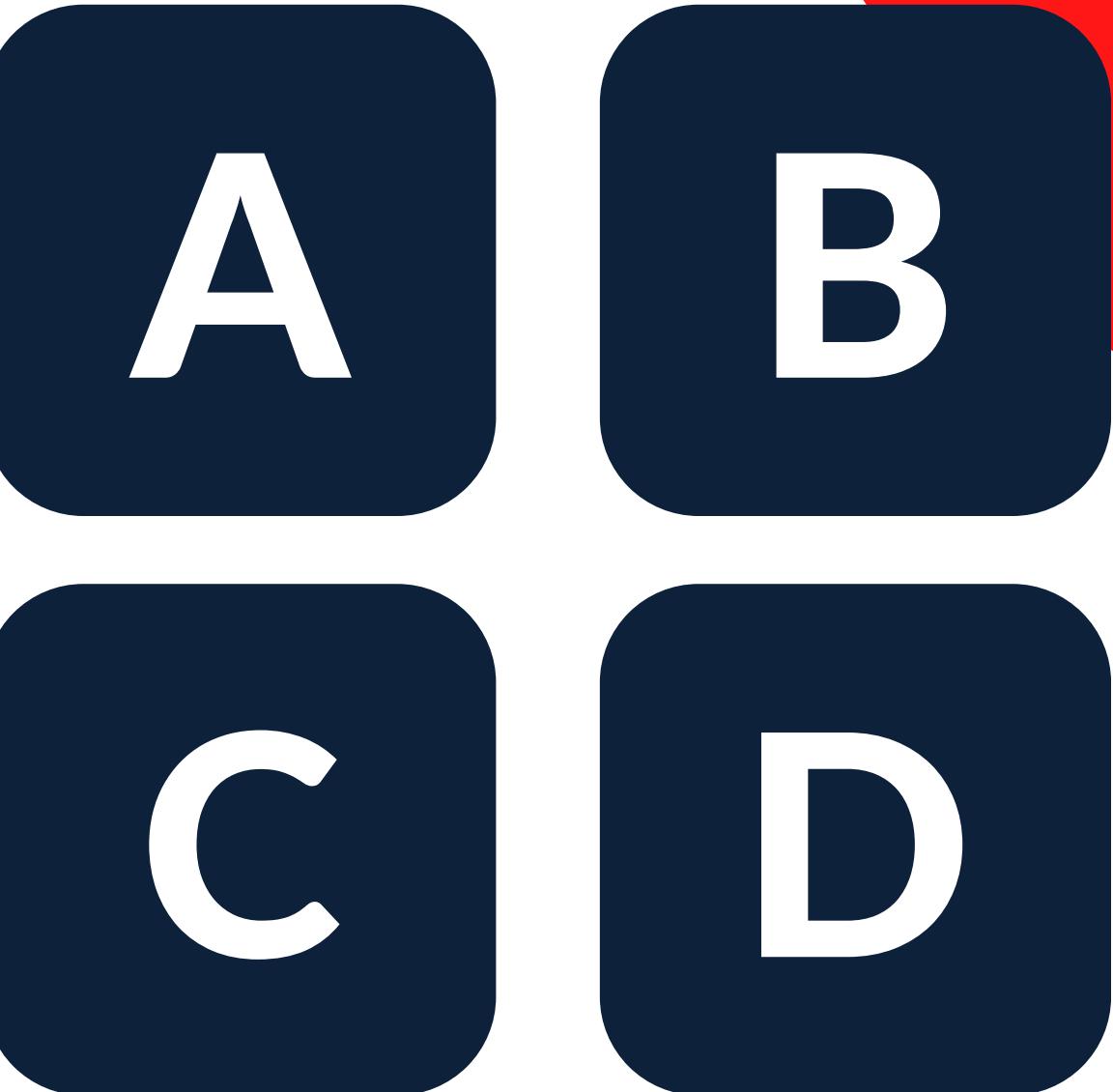
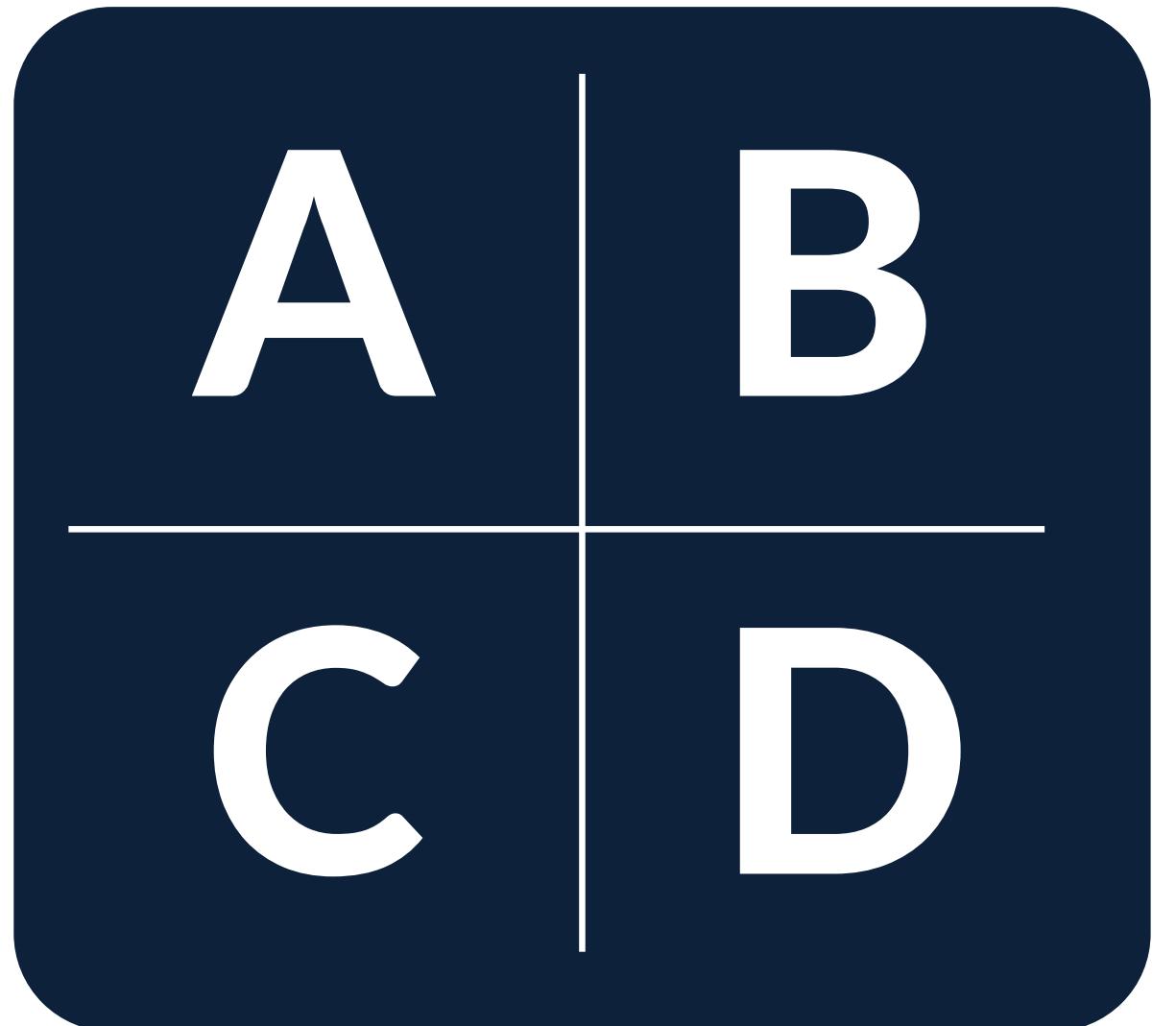
100MS

CONS



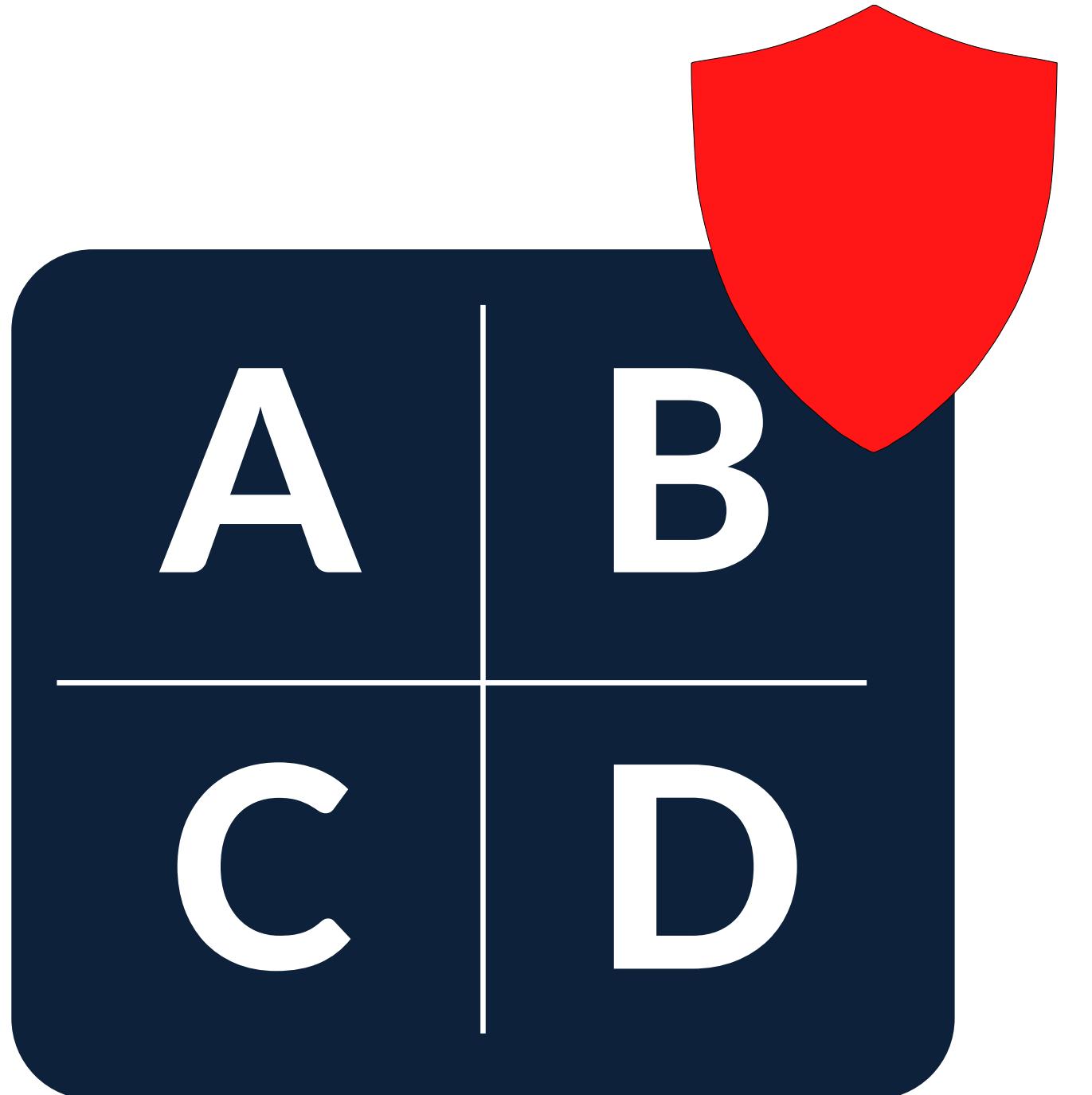
LATENCIA

¿Y si tenemos uno de los microservicios ubicado en otro continente?



SEGURIDAD

CONS

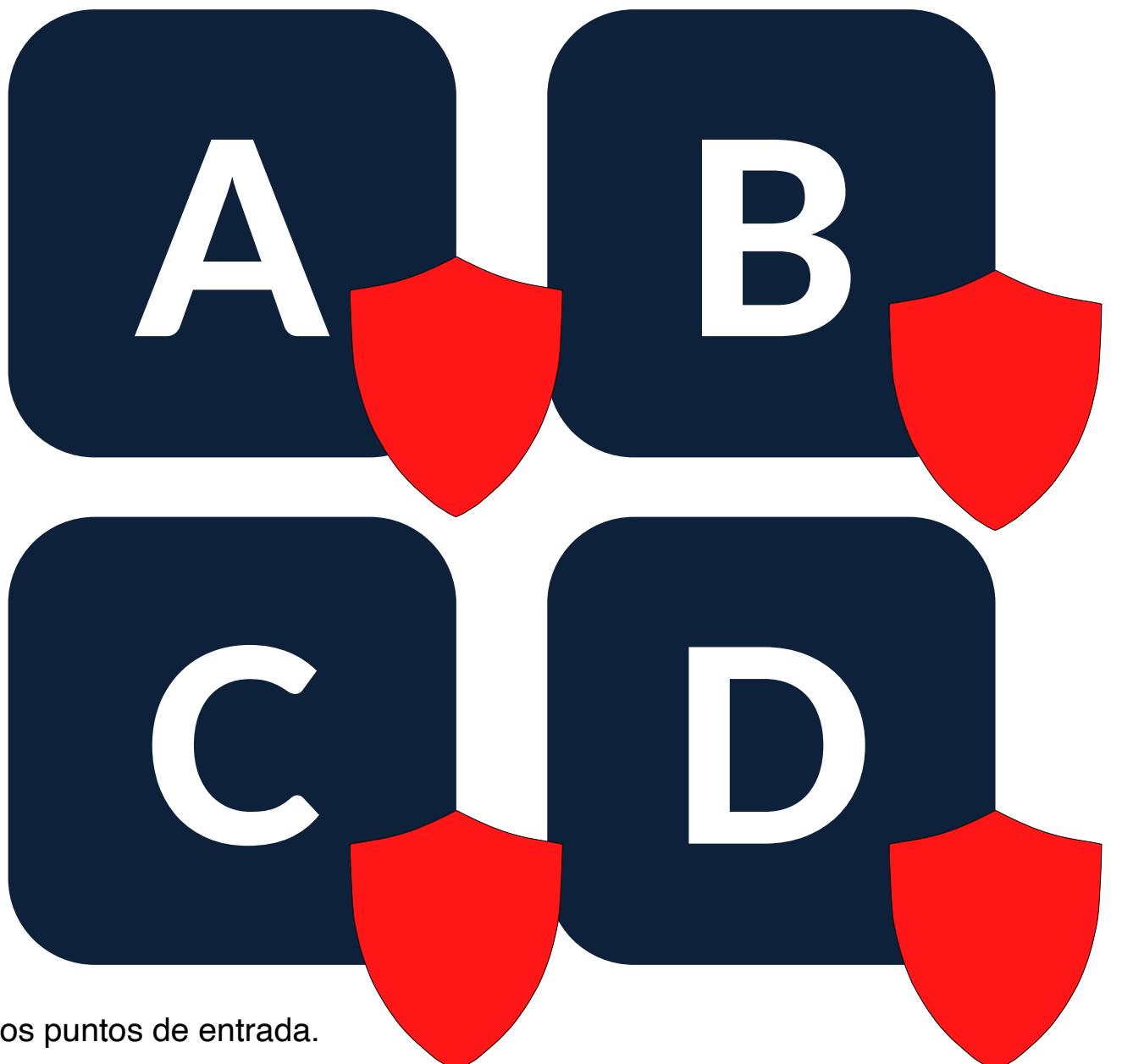


En un proyecto monolítico solo hay un punto de entrada expuesto al mundo exterior (Intranet, API...) para que la gente se conecte. Ahí prestaremos mayor atención (two factor authentication, tokens, lista negra, lista blanca...)

SEGURIDAD

CONS

CONS

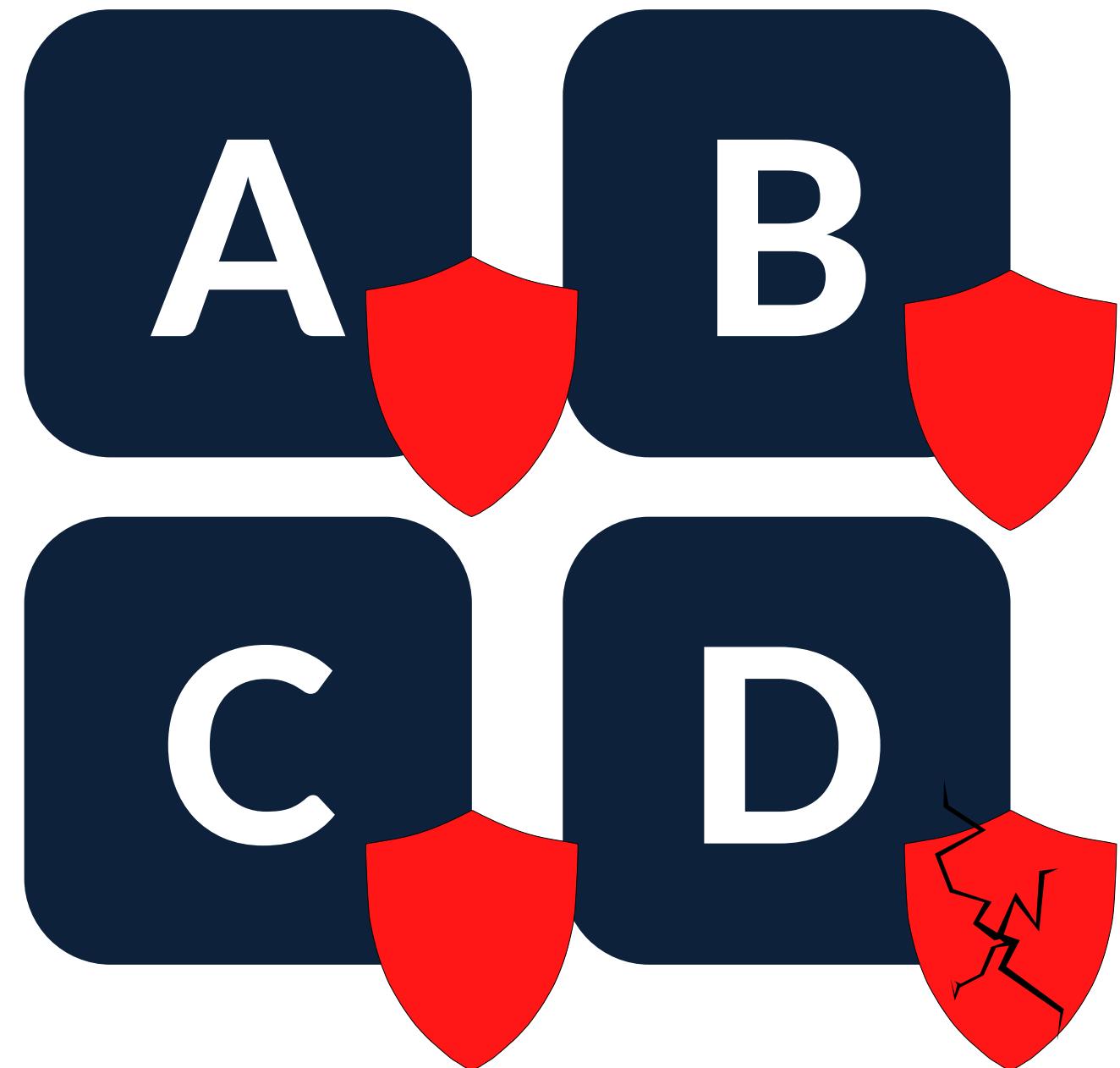


En un proyecto realizado con microservicios vamos a tener muchos puntos de entrada.
Cada microservicio tiene que ser responsable de su seguridad

SEGURIDAD

Mayores puntos de quiebre.

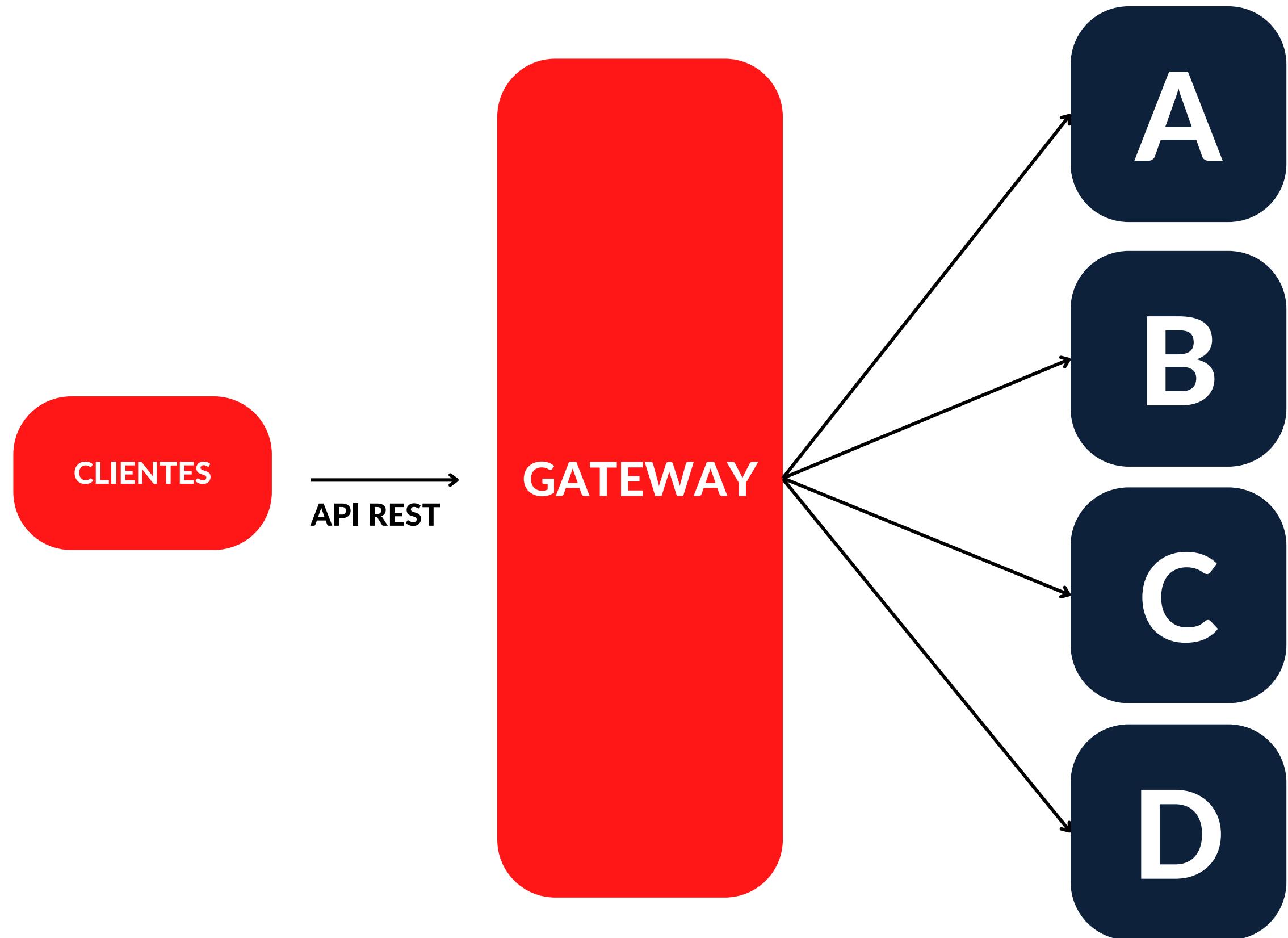
CONS



Si un microservicio falla, puede comprometer la seguridad de toda la aplicación, o comunicación entre todos ellos

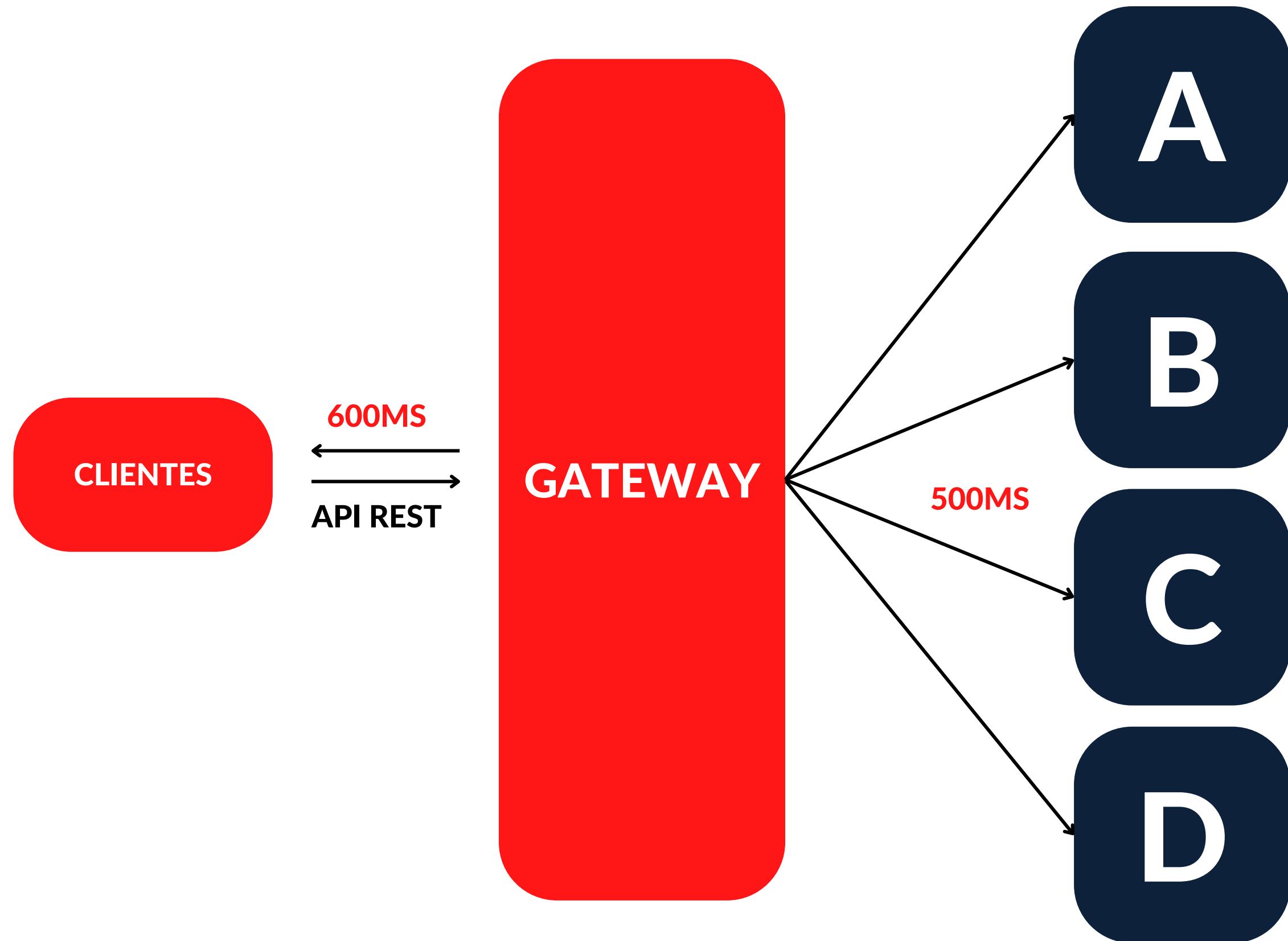
SEGURIDAD

CONS



MANEJO DE ERRORES Y PROBLEMAS

CONS



El usuario puede sentir que nuestra aplicación va lenta y es difícil rastrear el problema

Necesarios logs y reportes para ayudar a los administradores a saber qué está pasando

MANEJO DE ERRORES Y PROBLEMAS

TIPOS DE TRANSPORTE MICROSERVICIOS

Al final del día, todos tienen por objetivo enviar y/o recibir información entre ellos o entes externos.

- **HTTP/HTTPS:** Restful
- **gRPC:** Protocolo de buffers creado por Google
- **Queues o colas:** Mensajería tipo oficina postal, ejemplos como RabbitMQ, Apache Kafka y Amazon SQS.
- **Flujo de eventos (Streams):** Eventos pueden ser consumidos por multiples microservicios interesados. Ejemplos como Apache Kafka, Apache Pulsar y AWS Kinesis.
- **TCP/UDP:** Eficiente comunicación entre equipos en el cuarta capa del modelo OSI, confiable y ordenada.

BUENAS PRACTICAS MICROSERVICIOS

A

B

- Descomposición adecuada
- Responsabilidad única Cada microservicio una tarea y no más
- Tamaño adecuado



- **Independencia** Los microservicios deben ser aislados y crecer de forma independiente
- **Autonomía** Si B se cae, A debe seguir funcionando
- **Comunicación asíncrona**



Cada microservicio debe tener su BBDD independiente

Esto puede llevar a problemas de integridad referencial... que es el precio a pagar para tener esta arquitectura autónoma e independiente

Puede llevar a duplicidad, pero**

- **Escalamiento independiente** El crecimiento de A no debe afectar a B
- **Despliegues independientes** No puedo necesitar B para desplegar A



Duplicidad o problemas de integridad referencial, pero**

Integridad referencial == hijos huérfanos

- Tolerancia a errores
- Recuperación automática Esto puede hacerlo Kubernetes



- Seguridad independiente
- Autenticación, autorización, cifrado

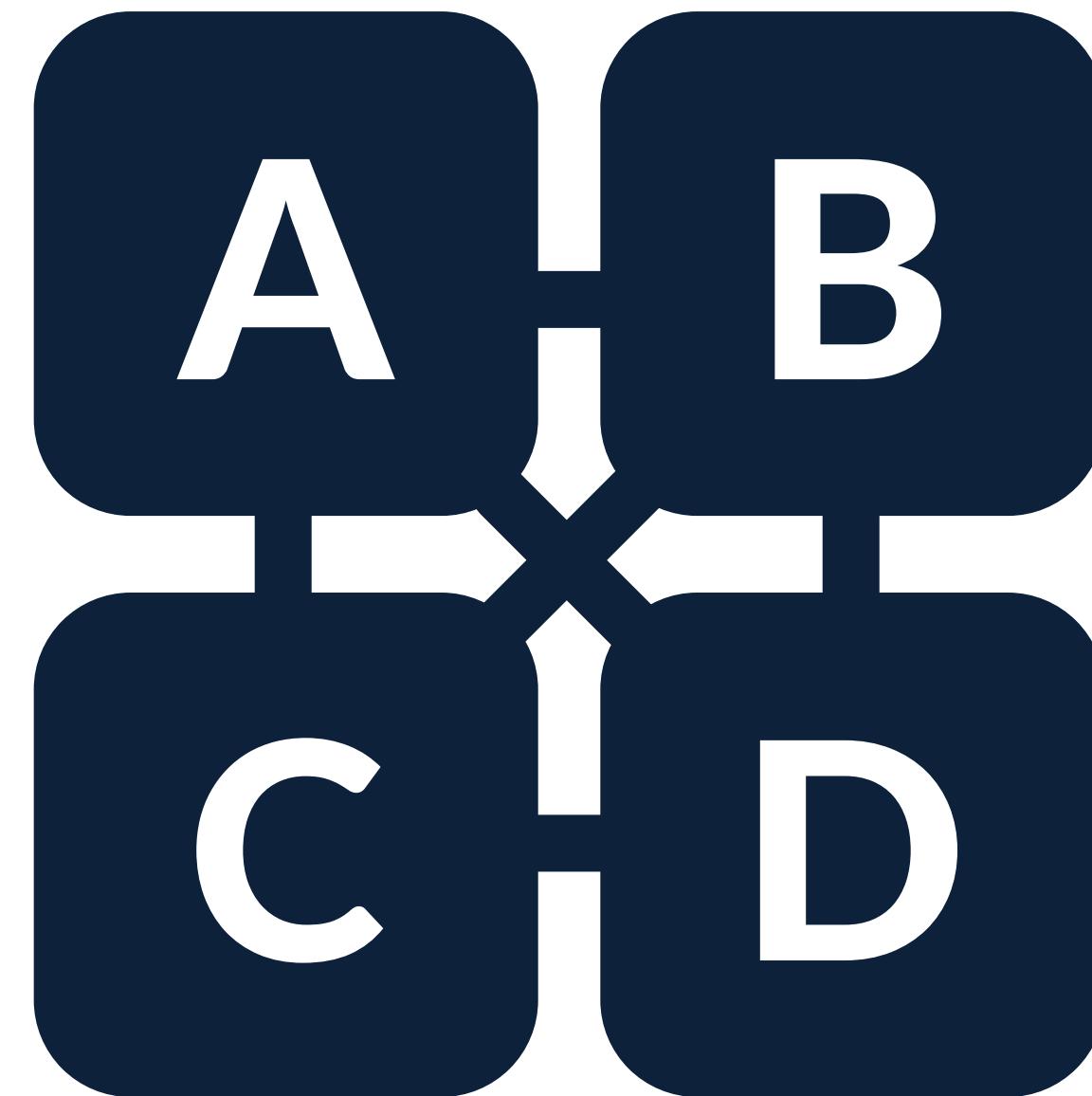
Con tokens, por ejemplo



PERO *

- **Evitar el acoplamiento excesivo.**
- **Evitar la dependencia entre ellos.**

Tienen que comunicarse pero hay que evitar el exceso, por ejemplo, que todos llamen continuamente a uno de ellos





SUFICIENTE TEORÍA
EMPEZEMOS LA PRÁCTICA

Objetivo

Postman, Insomnia, App Web...



API REST



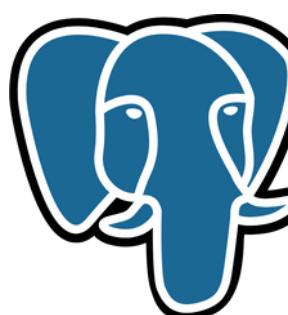
TCP



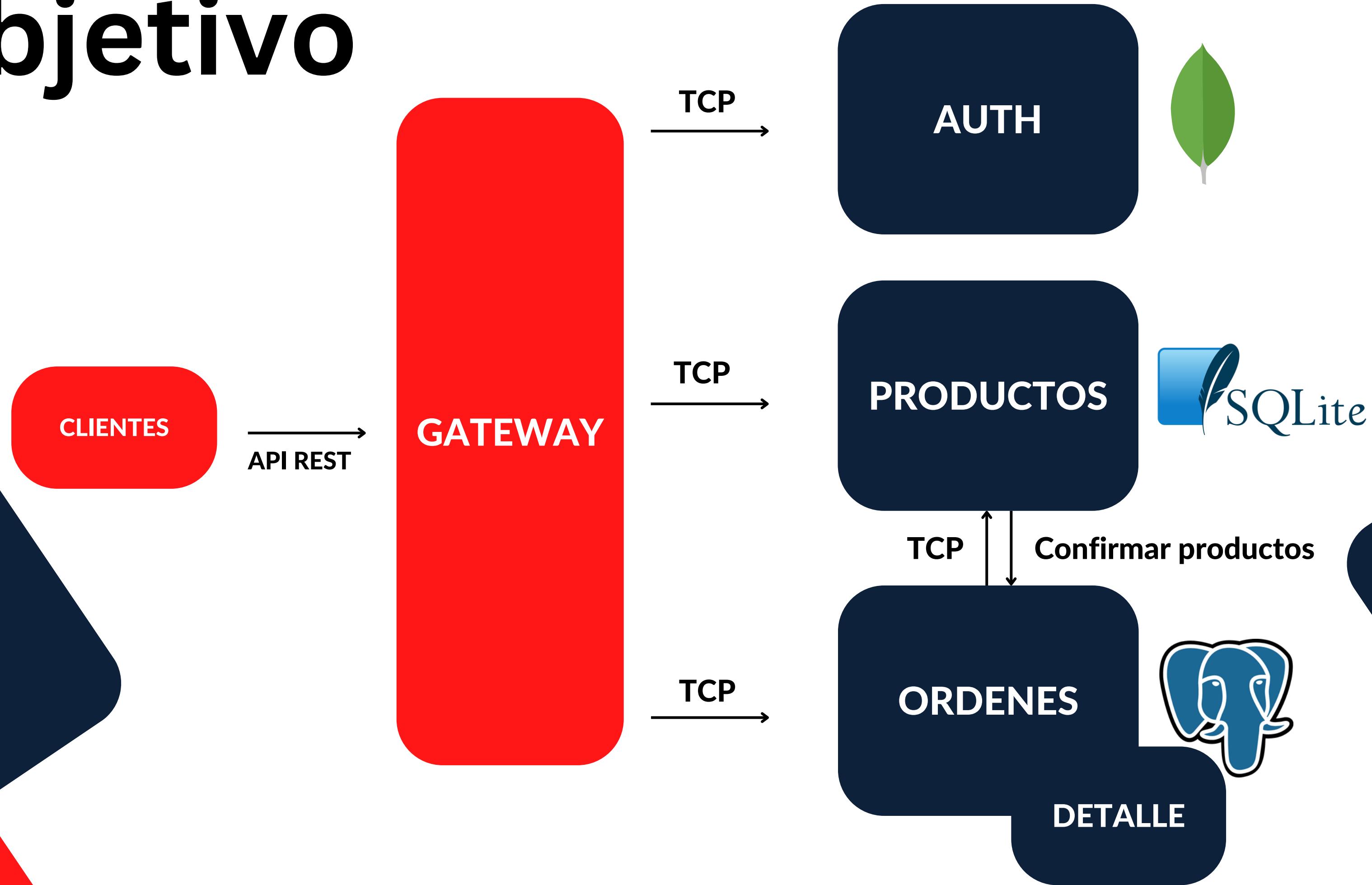
TCP



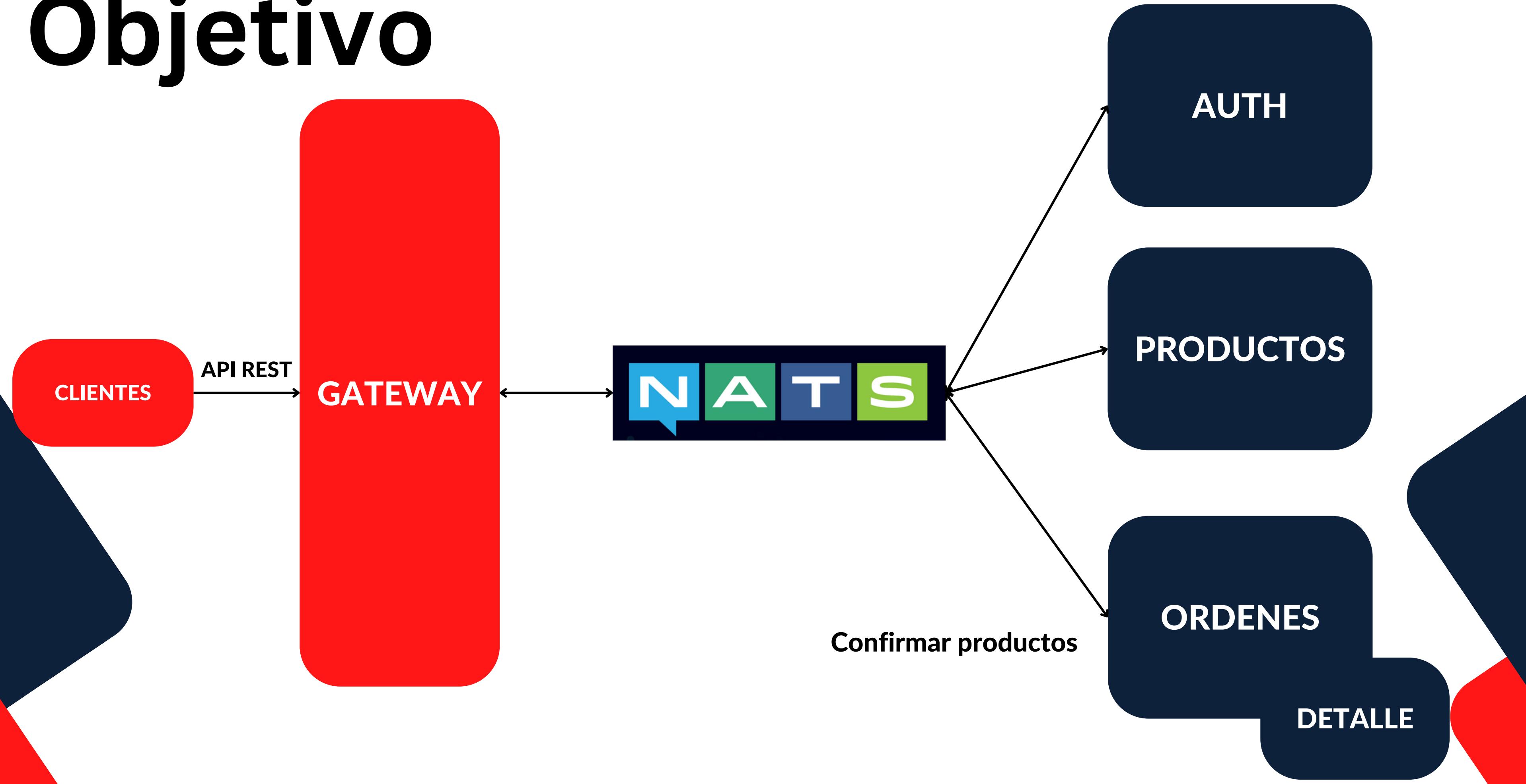
TCP



Objetivo



Objetivo

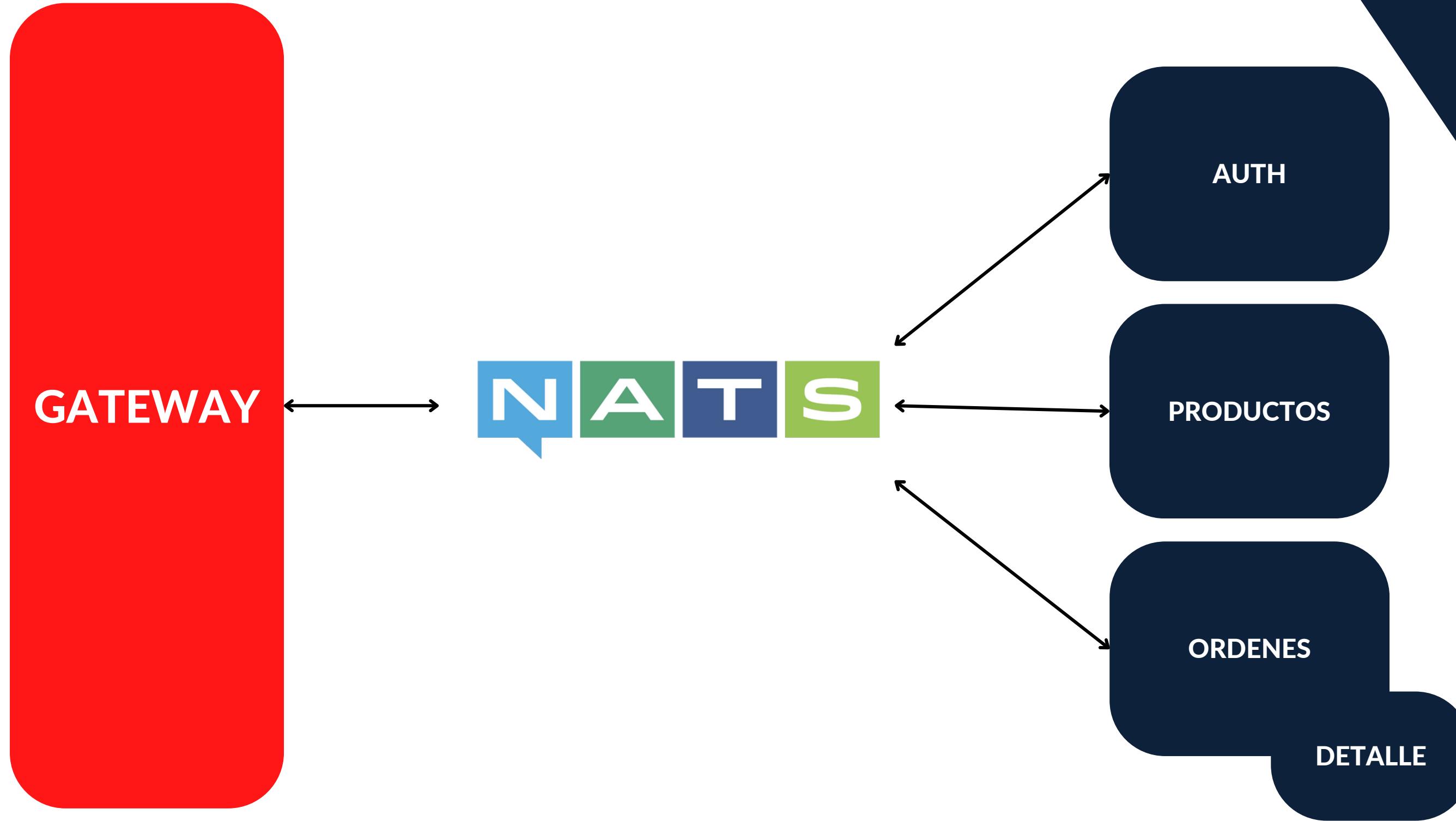


NATS BROKER



Open source

Middleman



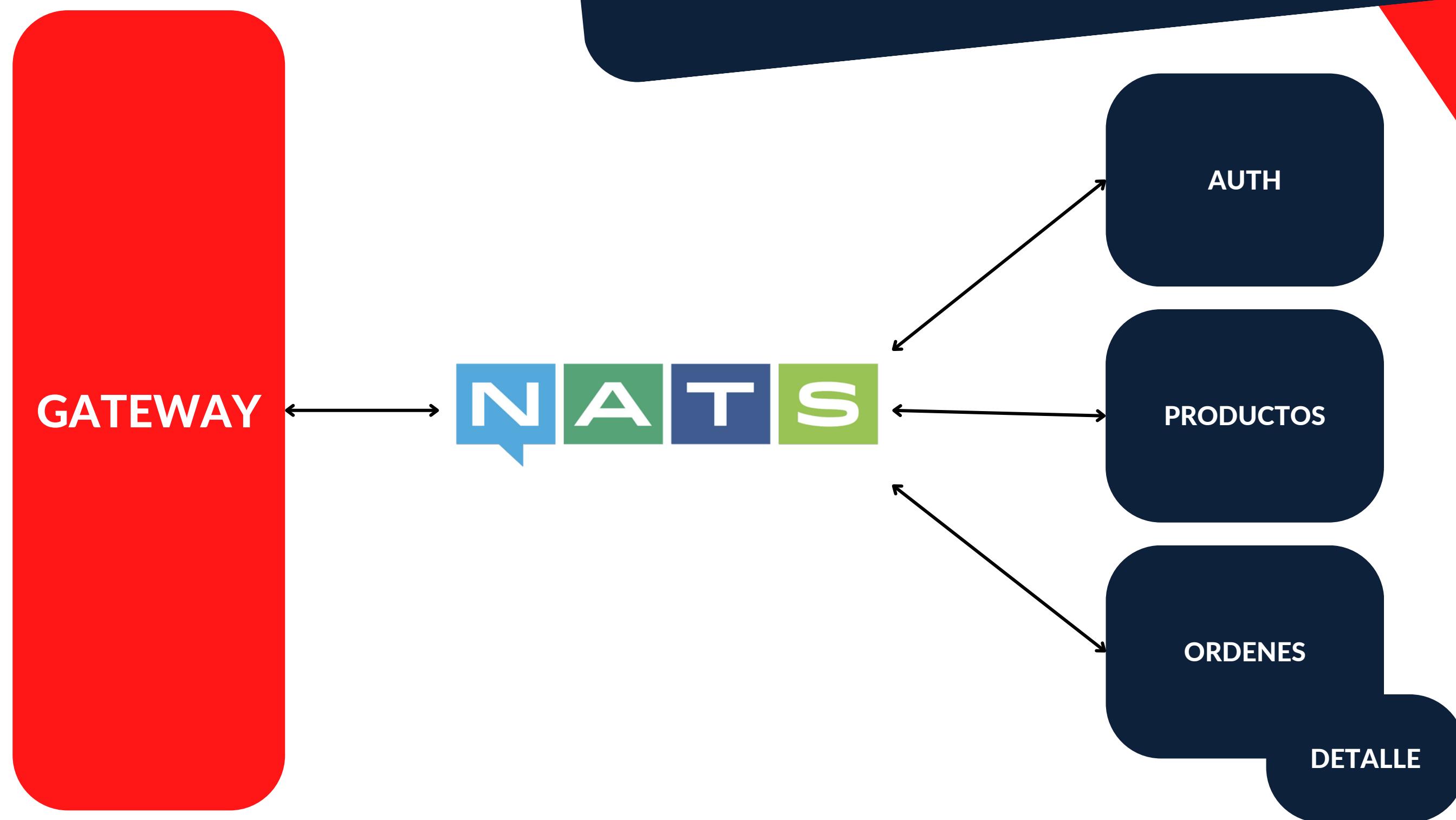
UNA TELA QUE UNE TODOS NUESTROS SISTEMAS DISTRIBUÍDOS

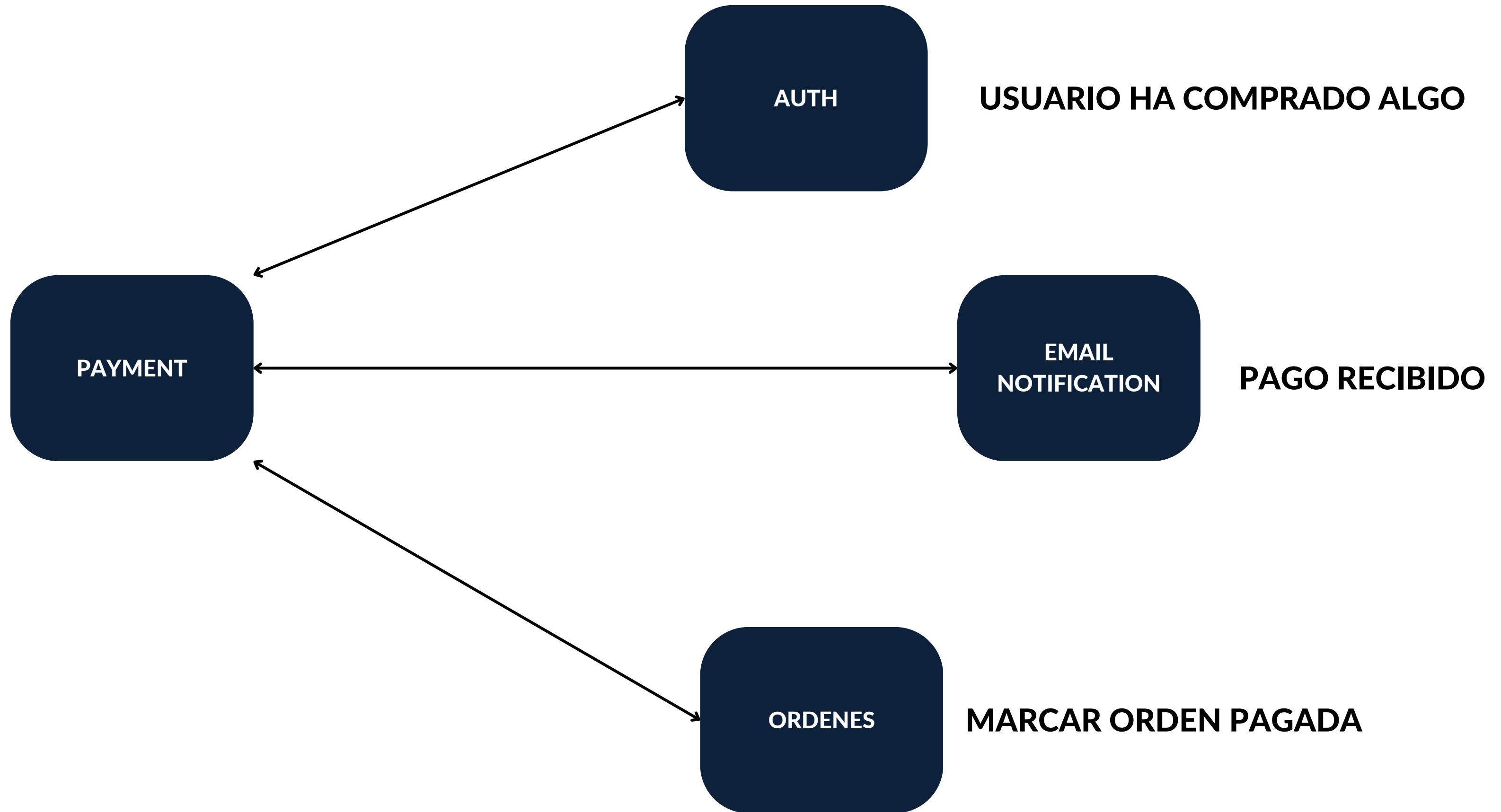


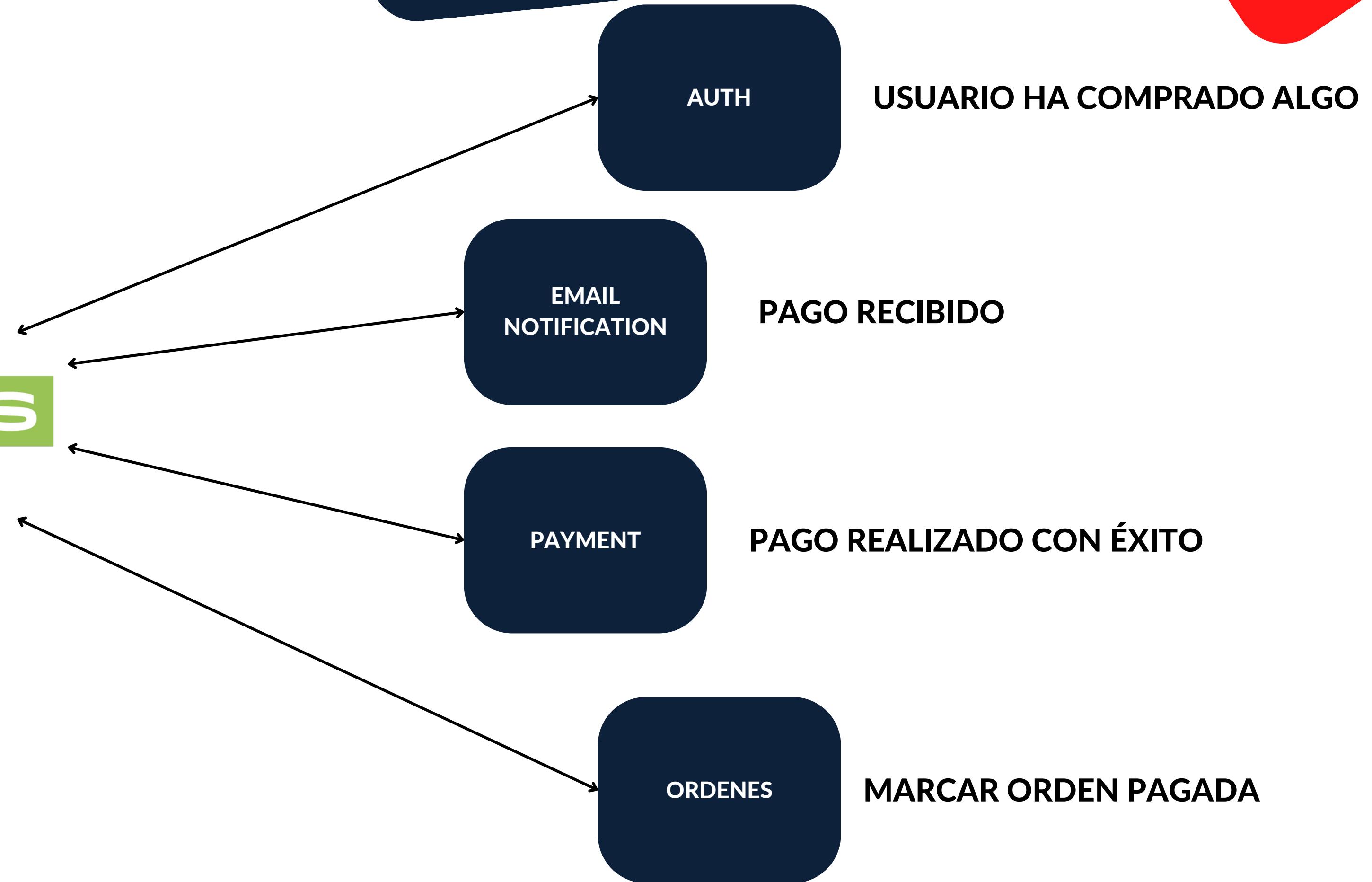
- Trabaja con mensajería tipo publicar y suscribir.
- Hay temas (topics/subjects) a los cuales se escucha.
- Puede tener múltiples escuchas (listeners) al mismo topic.
- Pensado en para escalamiento horizontal.
- Seguridad, balanceo de carga incluído.
- Payload agnóstico.
- Rápido y eficiente.

Conectar microservicios

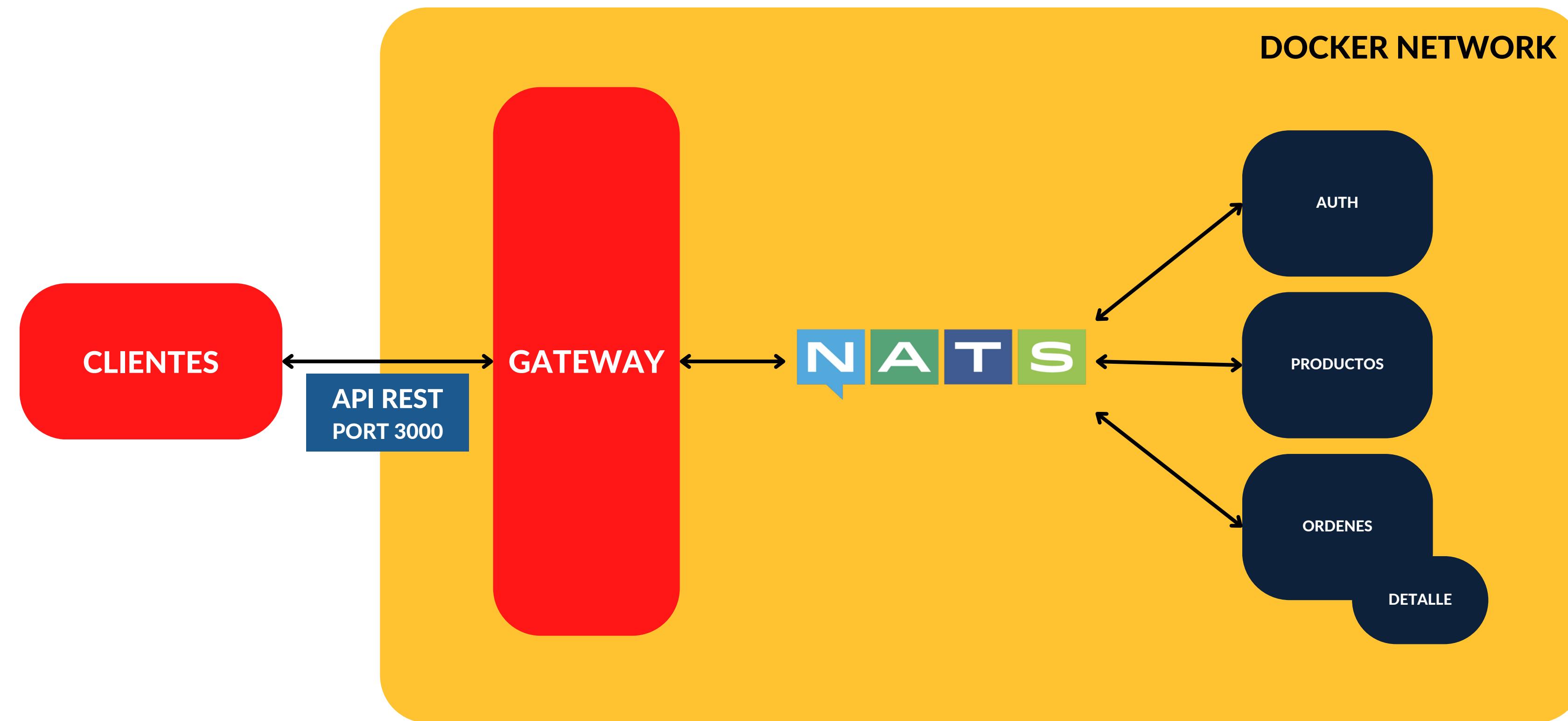
DE FORMA EFICIENTE Y RÁPIDA







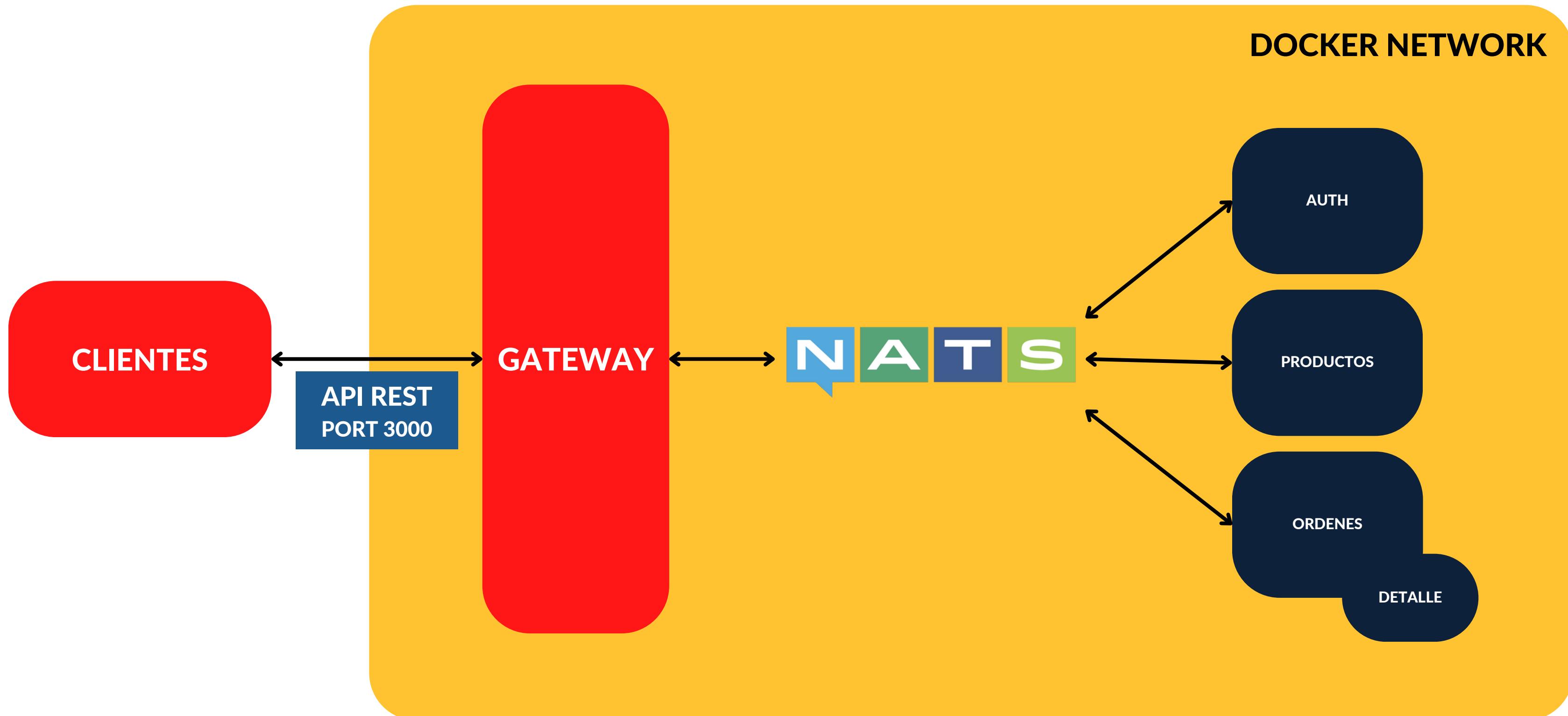
Docker Network



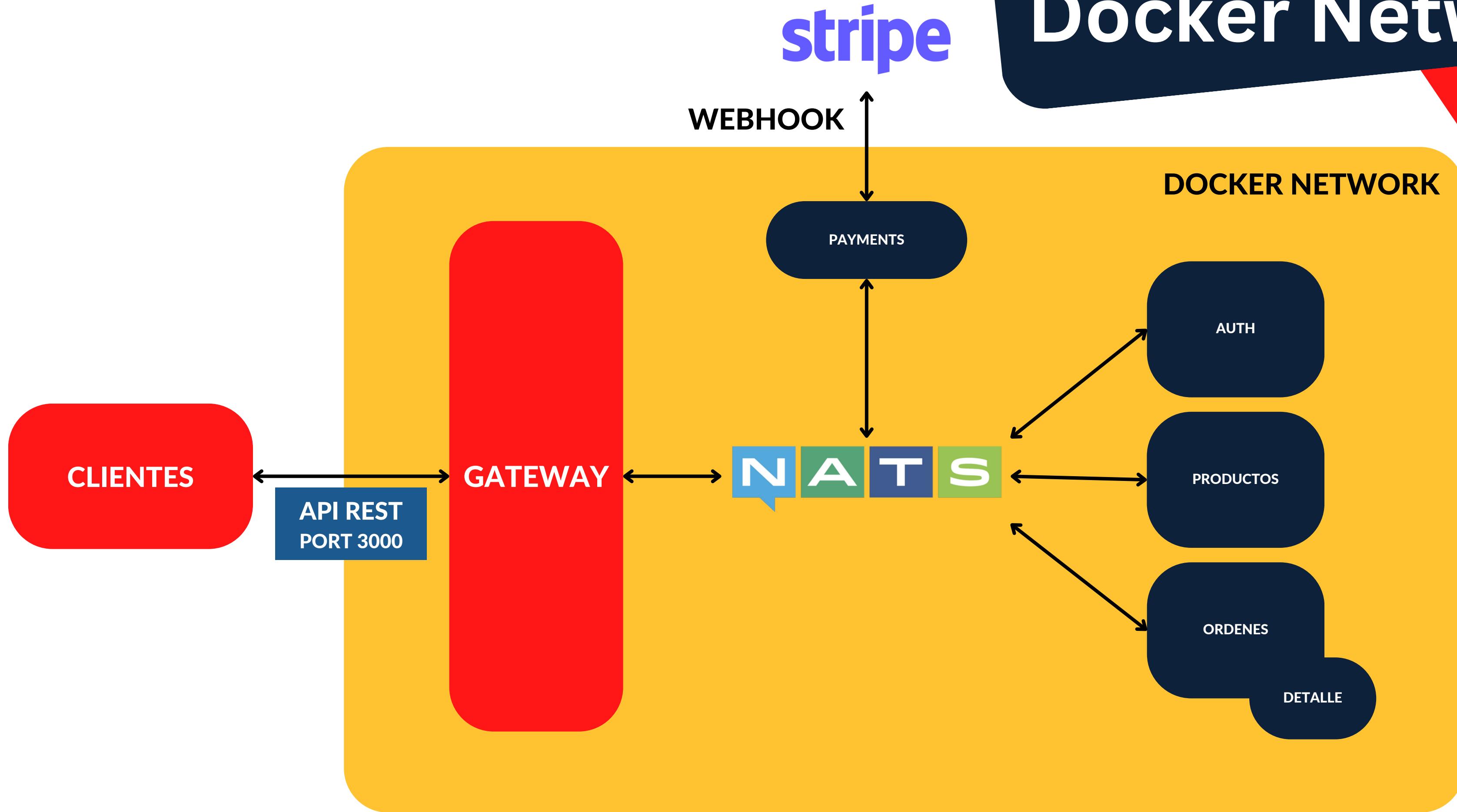
Docker Network

PAYMENTS

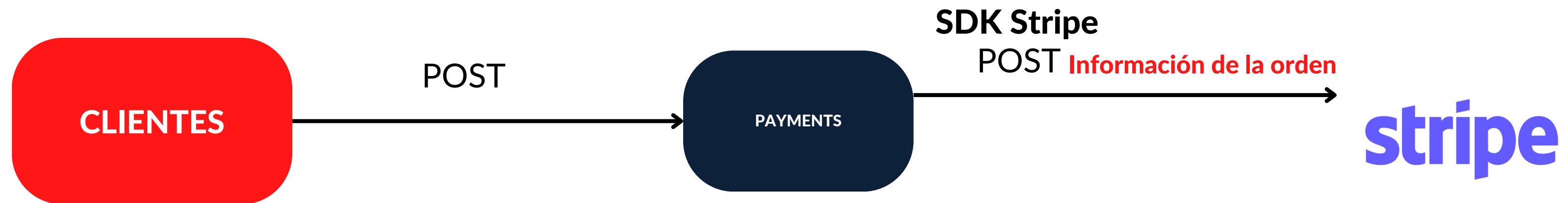
- POST: Crear sesión de pago
- Webhook: Escuchar pago desde Stripe



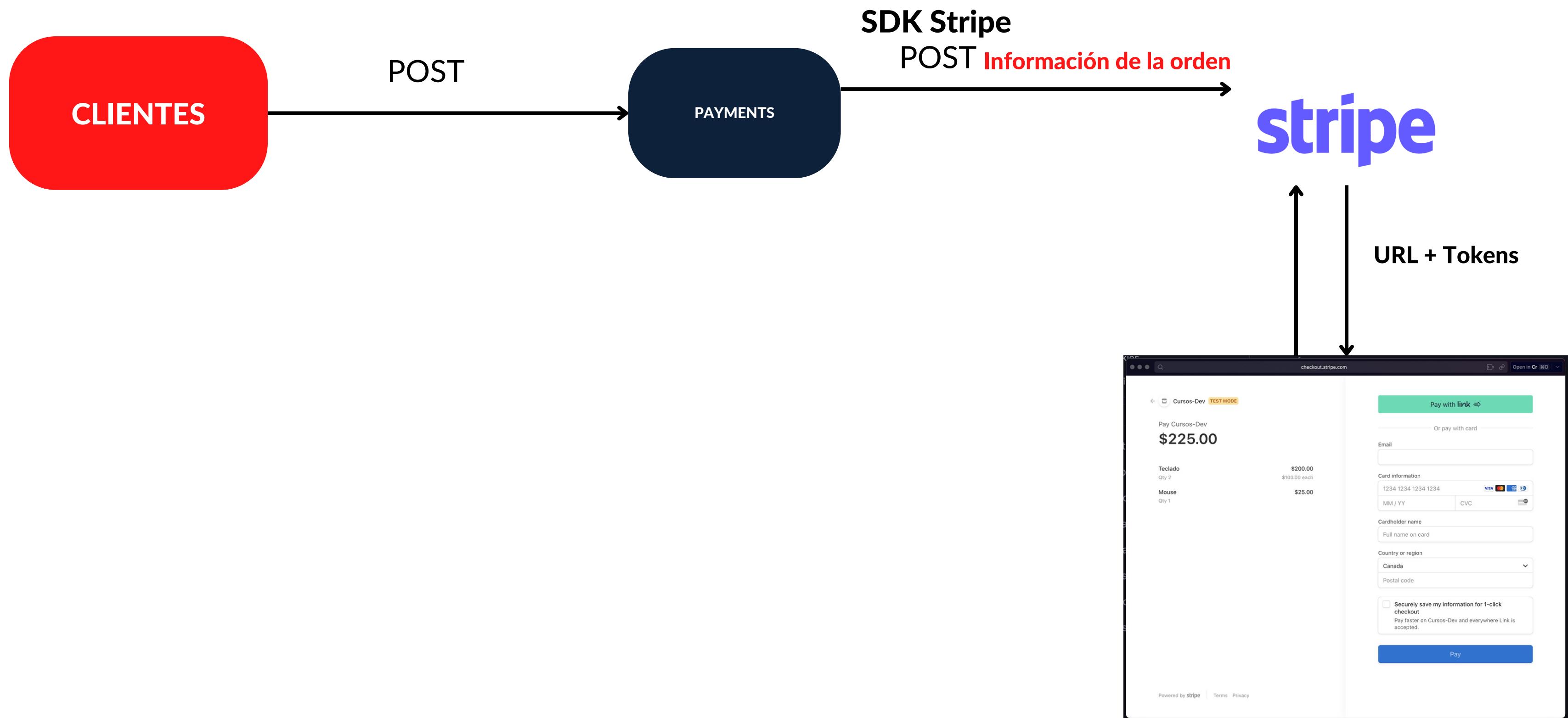
Docker Network



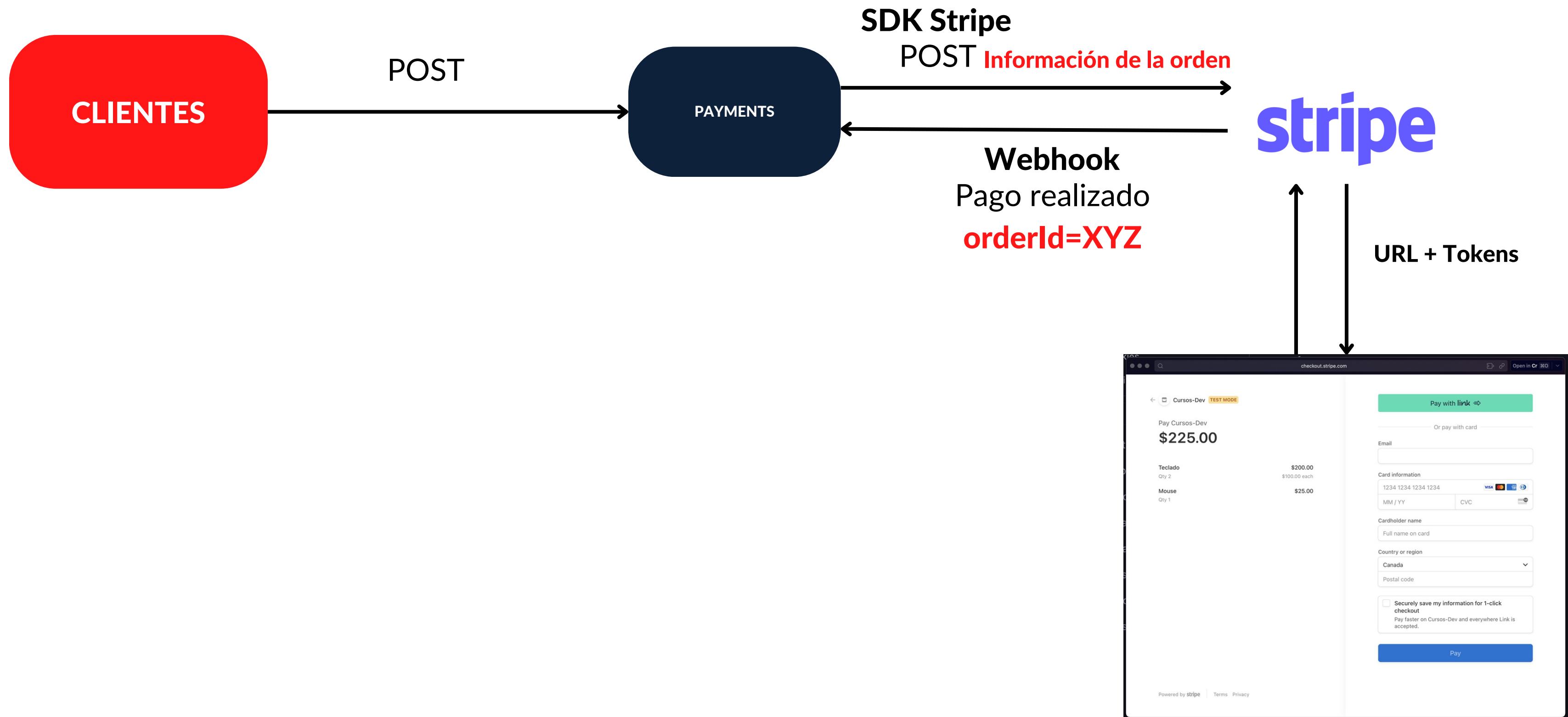
stripe



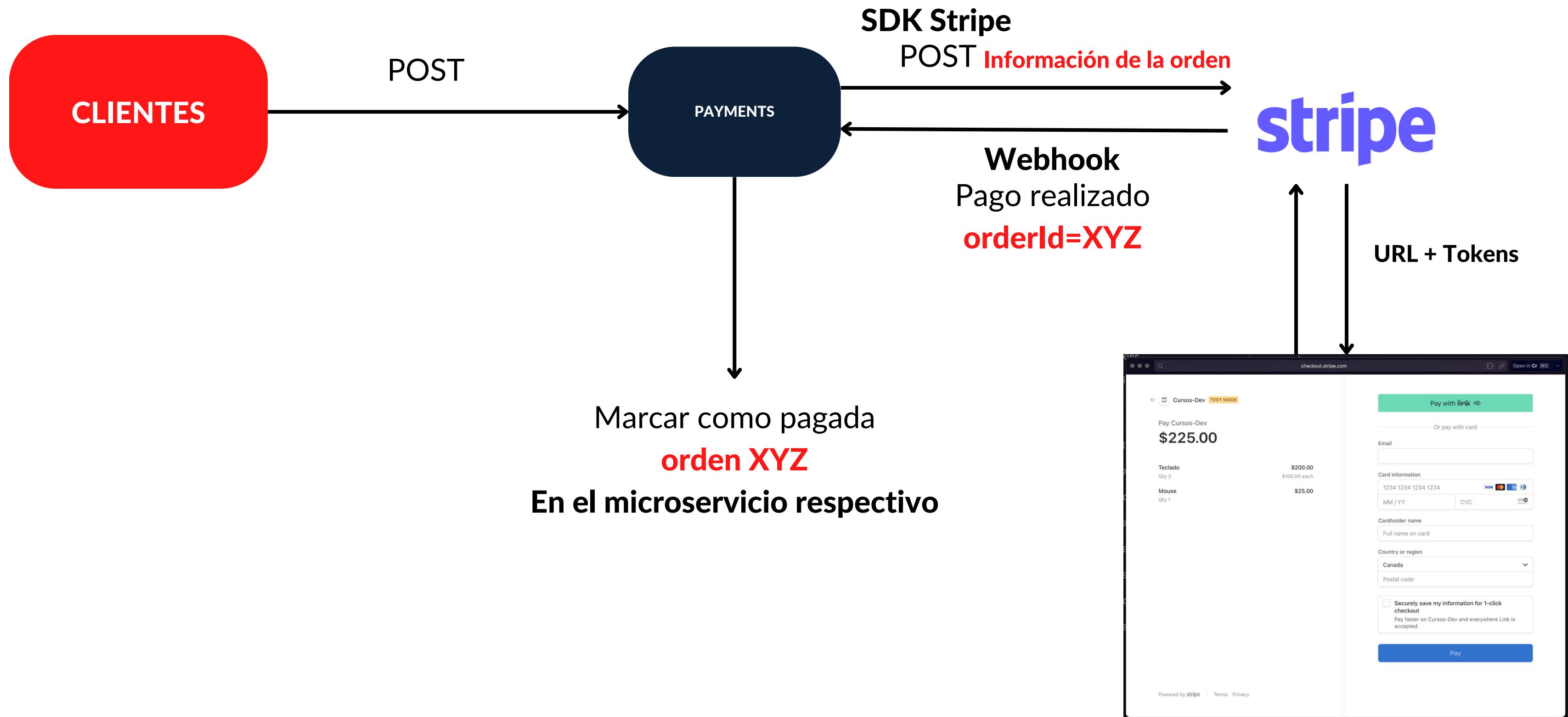
stripe



stripe



stripe



KUBERNETES

INTRODUCCIÓN

Permite a los desarrolladores desplegar, actualizar y administrar aplicaciones de manera eficiente, proporcionando herramientas para la gestión de recursos, el balanceo de carga, la auto curación y la escalabilidad horizontal



KUBERNETES



KUBERNETES



KUBERNETES



KUBERNETES



REPLICAS



CLUSTER

MASTER NODE

API SERVER
Administración

**CONTROLLER
MANAGER**
Sabe lo que pasa

SCHEDULER
Asegura el cambio de PODs

Controla toda la infraestructura

- Es el más importante
- Puede tener réplicas

Accedemos a él

- CLI
- UI
- API

ETCD
KeyValue storage



CLUSTER



Controla toda la infraestructura

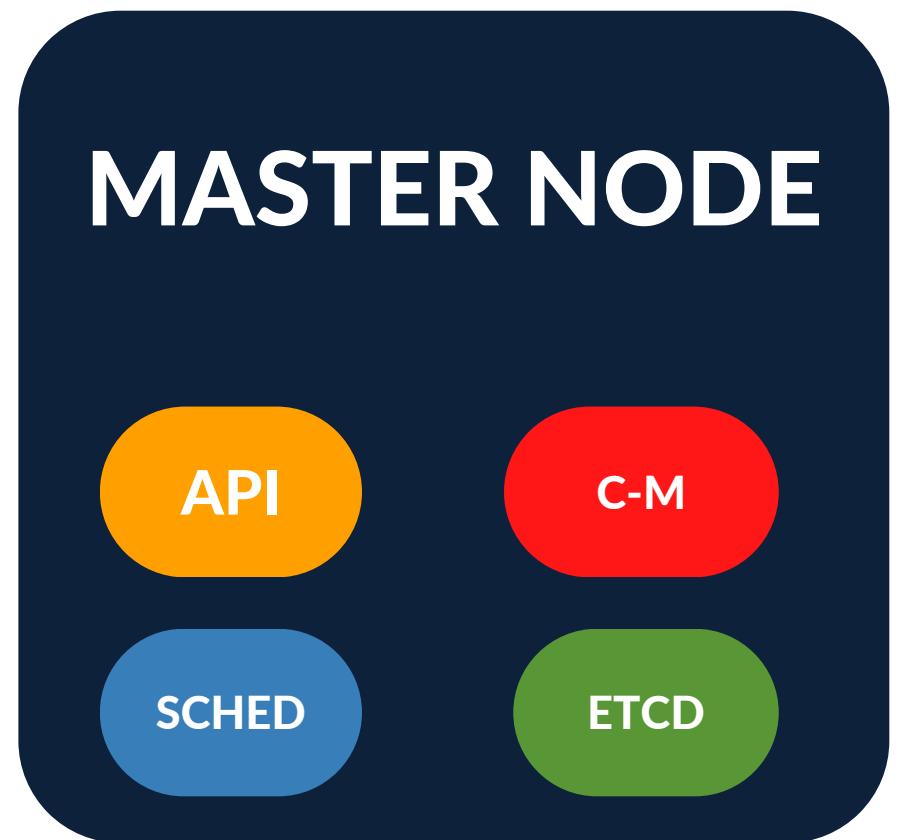
- Es el más importante
- Puede tener réplicas

Accedemos a él

- CLI
- UI
- API

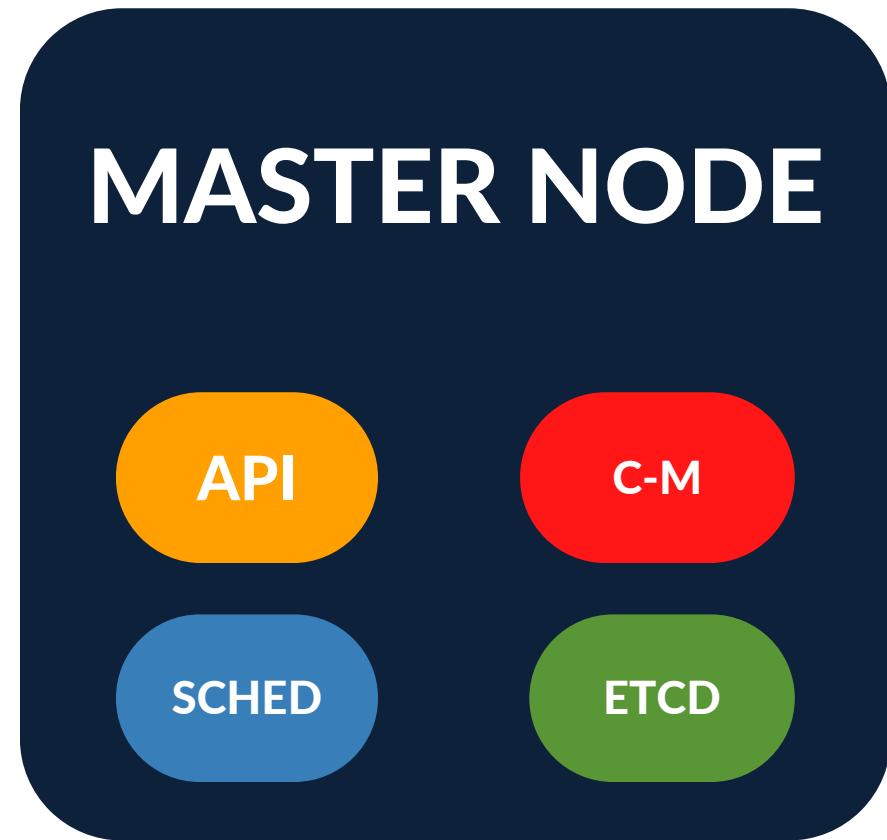
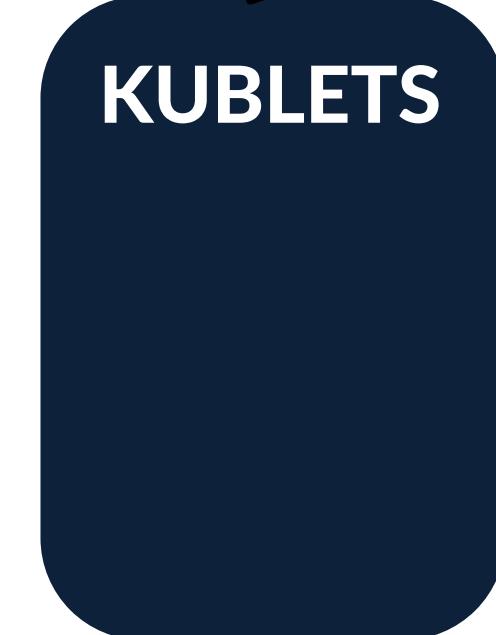
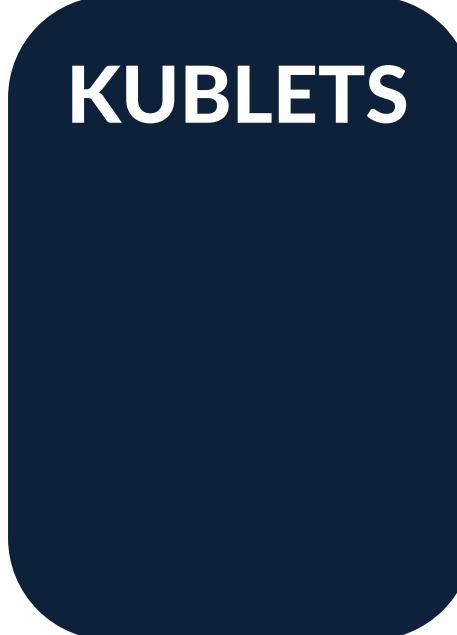


CLUSTER



CLUSTER

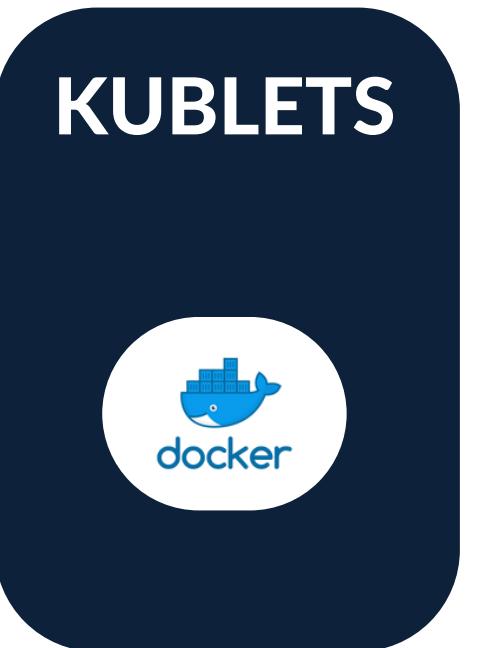
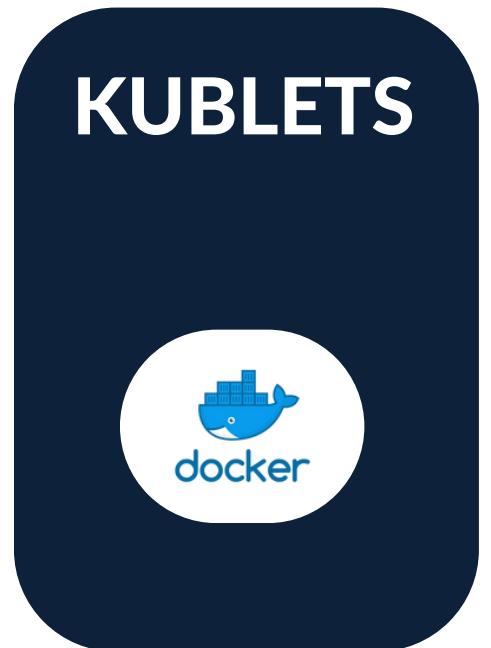
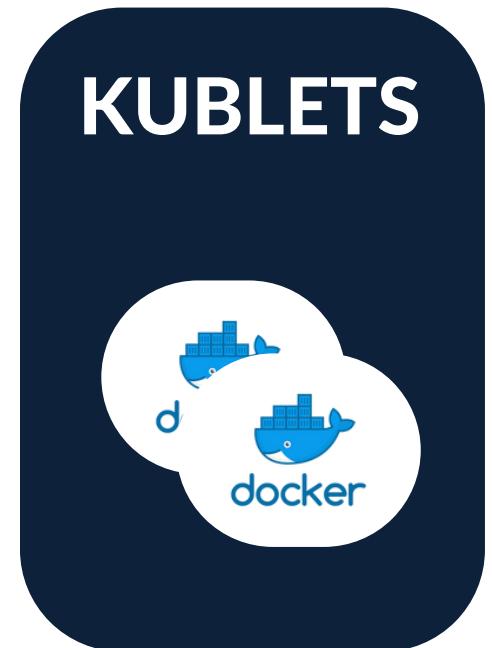
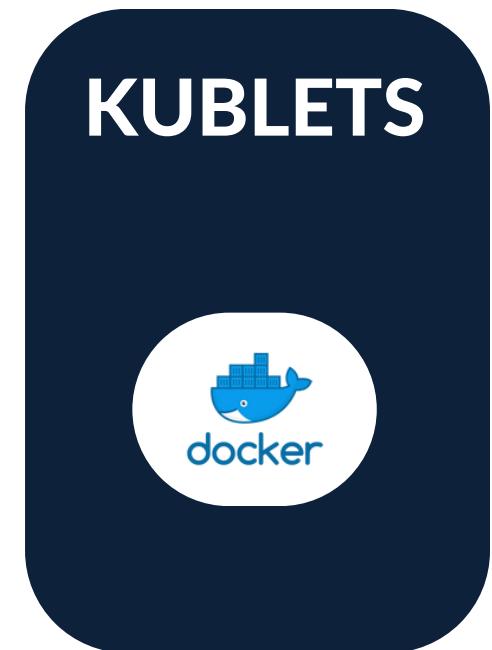
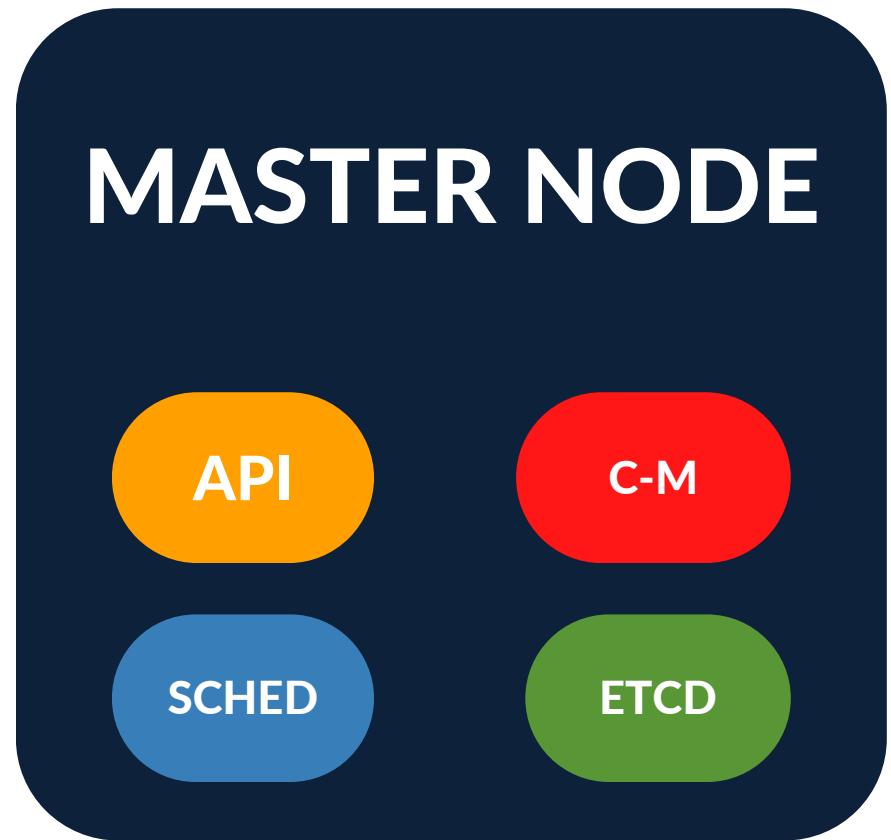
WORKER NODES



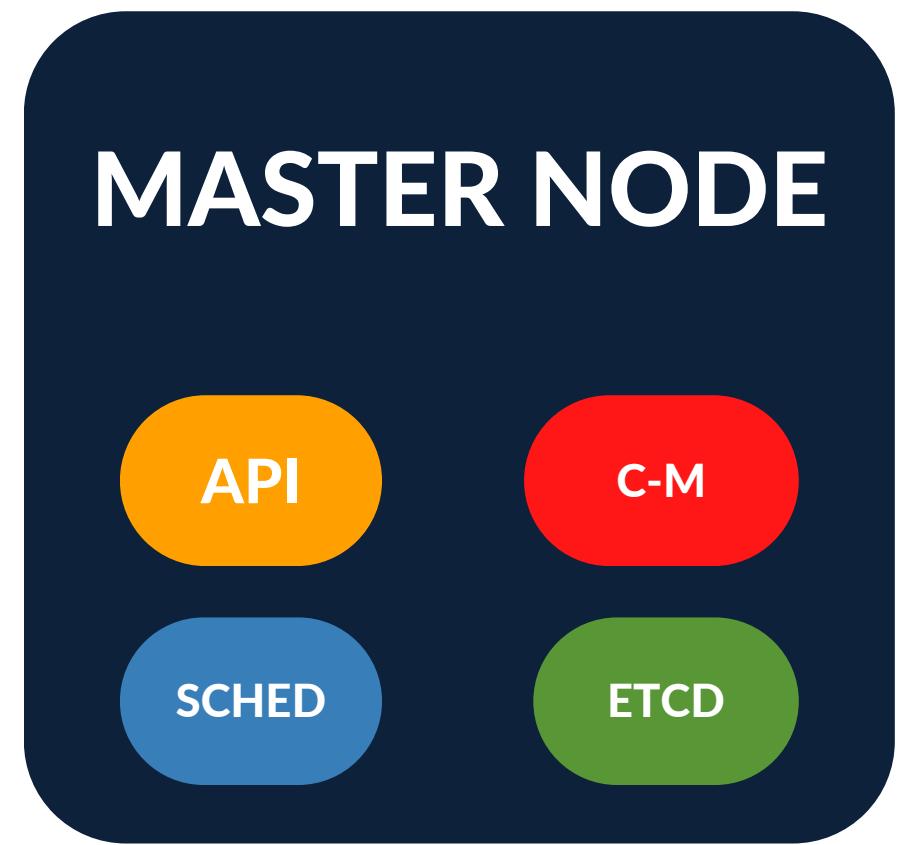
Kublet: Proceso corriendo que permite la comunicación entre el cluster



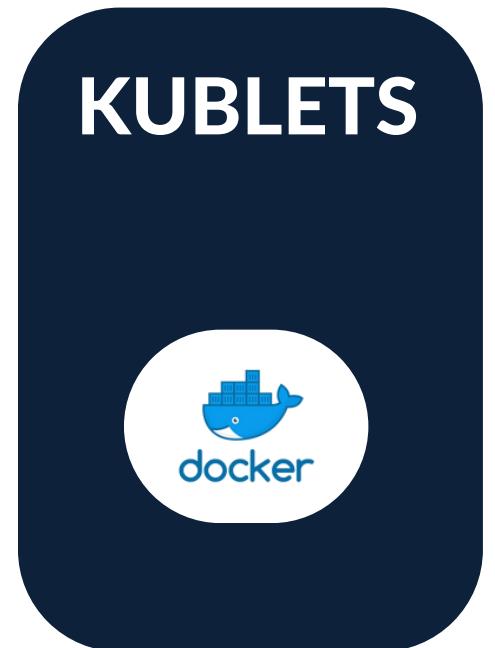
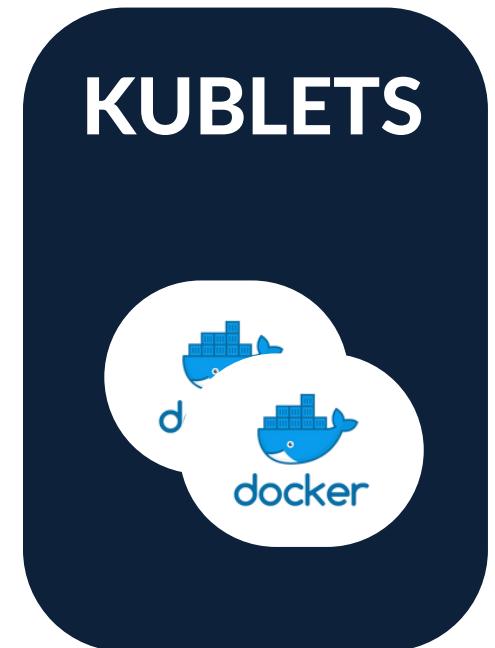
CLUSTER



CLUSTER



RED VIRTUAL

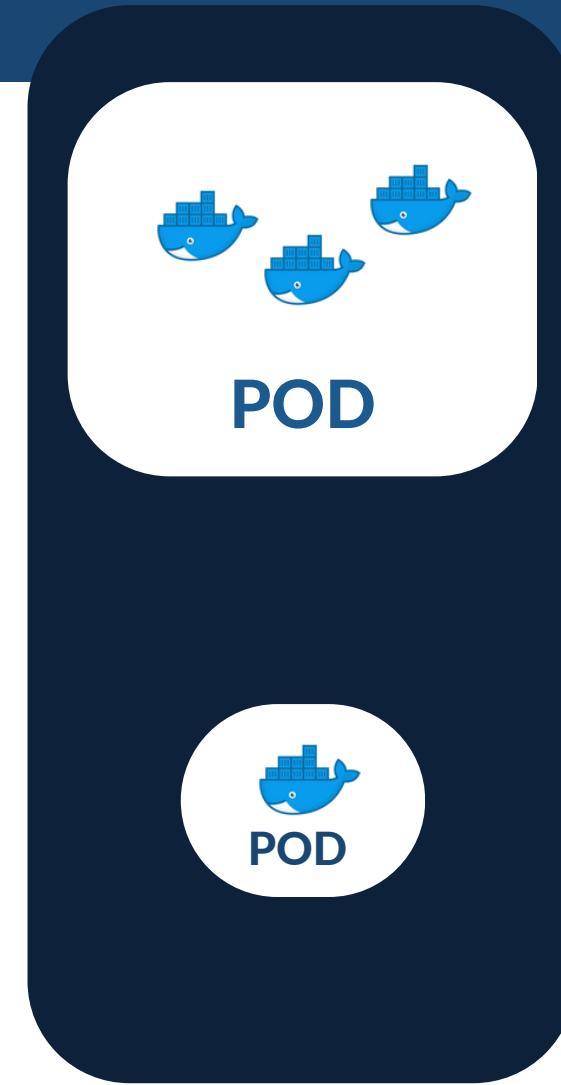
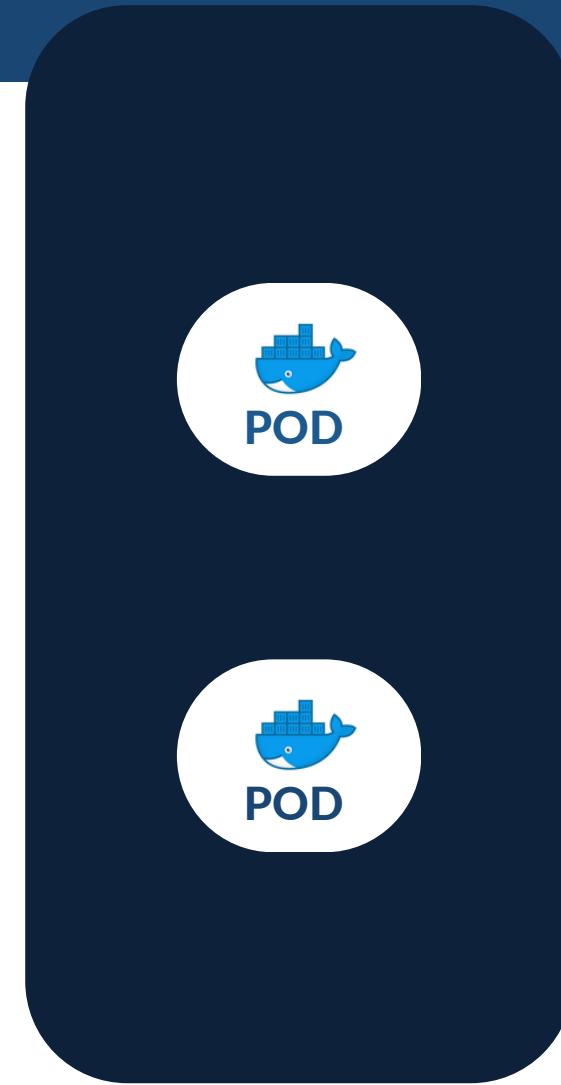


Virtual Network: Convierte nuestro cluster en una única máquina



PODs

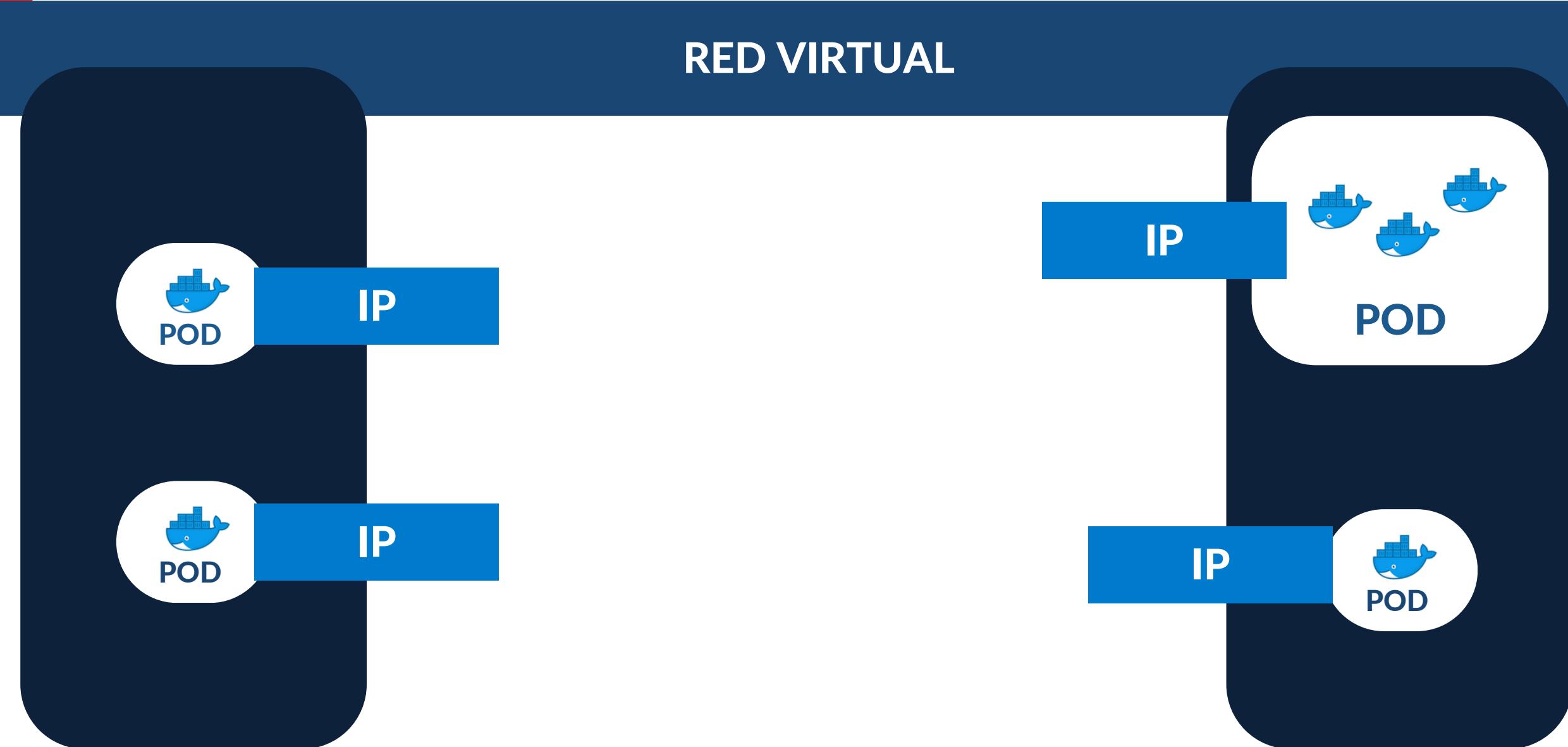
RED VIRTUAL



POD: Unidad más pequeña, es efímera, con su propia IP estática e ID único



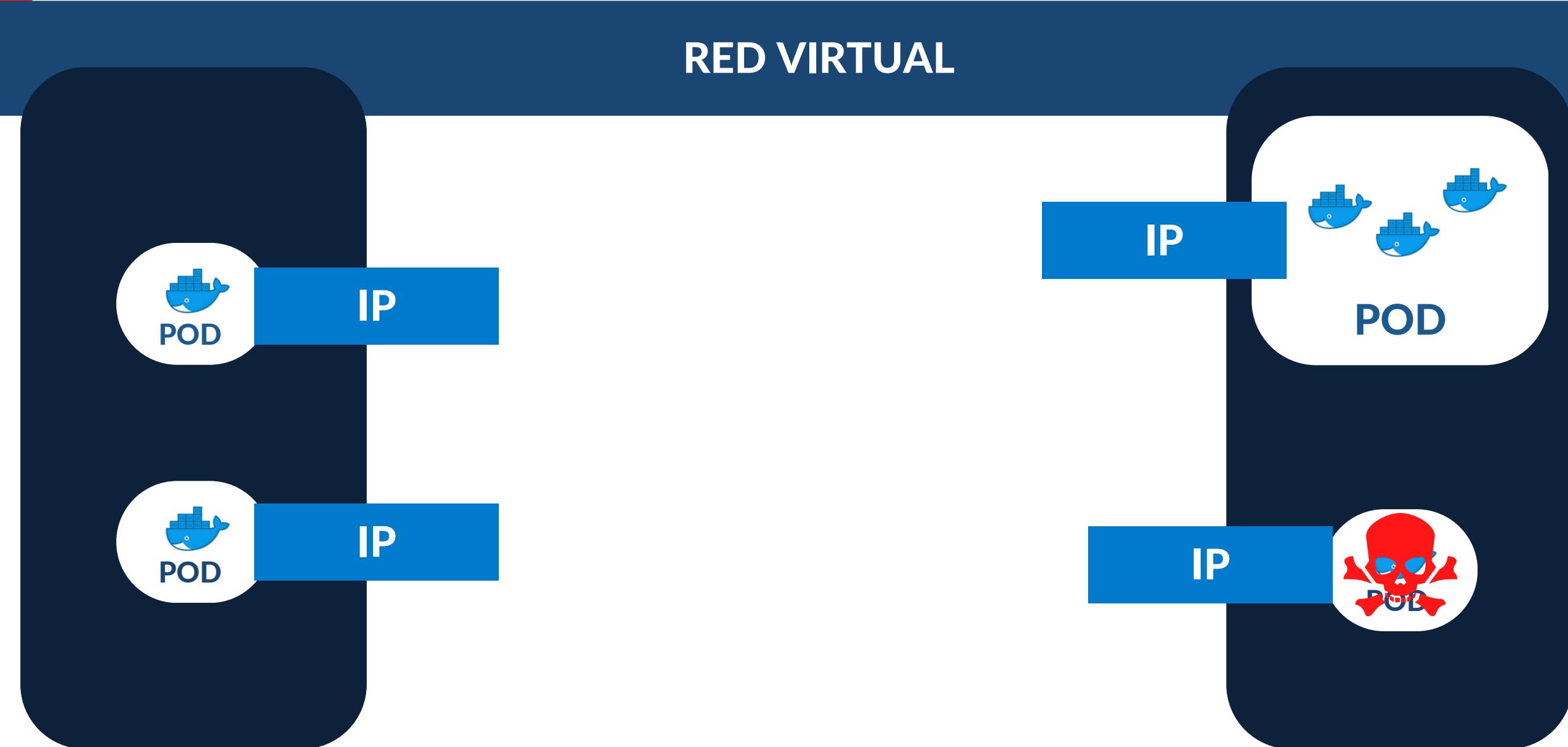
PODs



POD: Unidad más pequeña, es efímera, con su propia IP estática e ID único



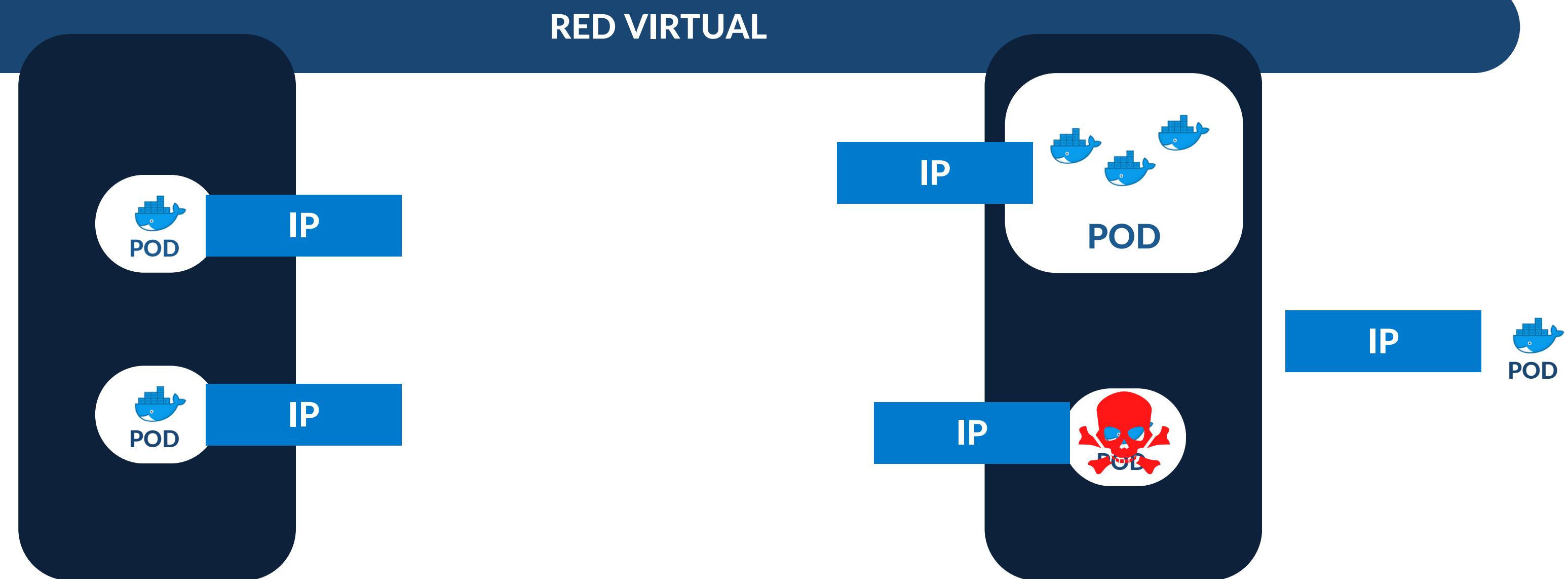
PODs



POD: Unidad más pequeña, es efímera, con su propia IP estática e ID único



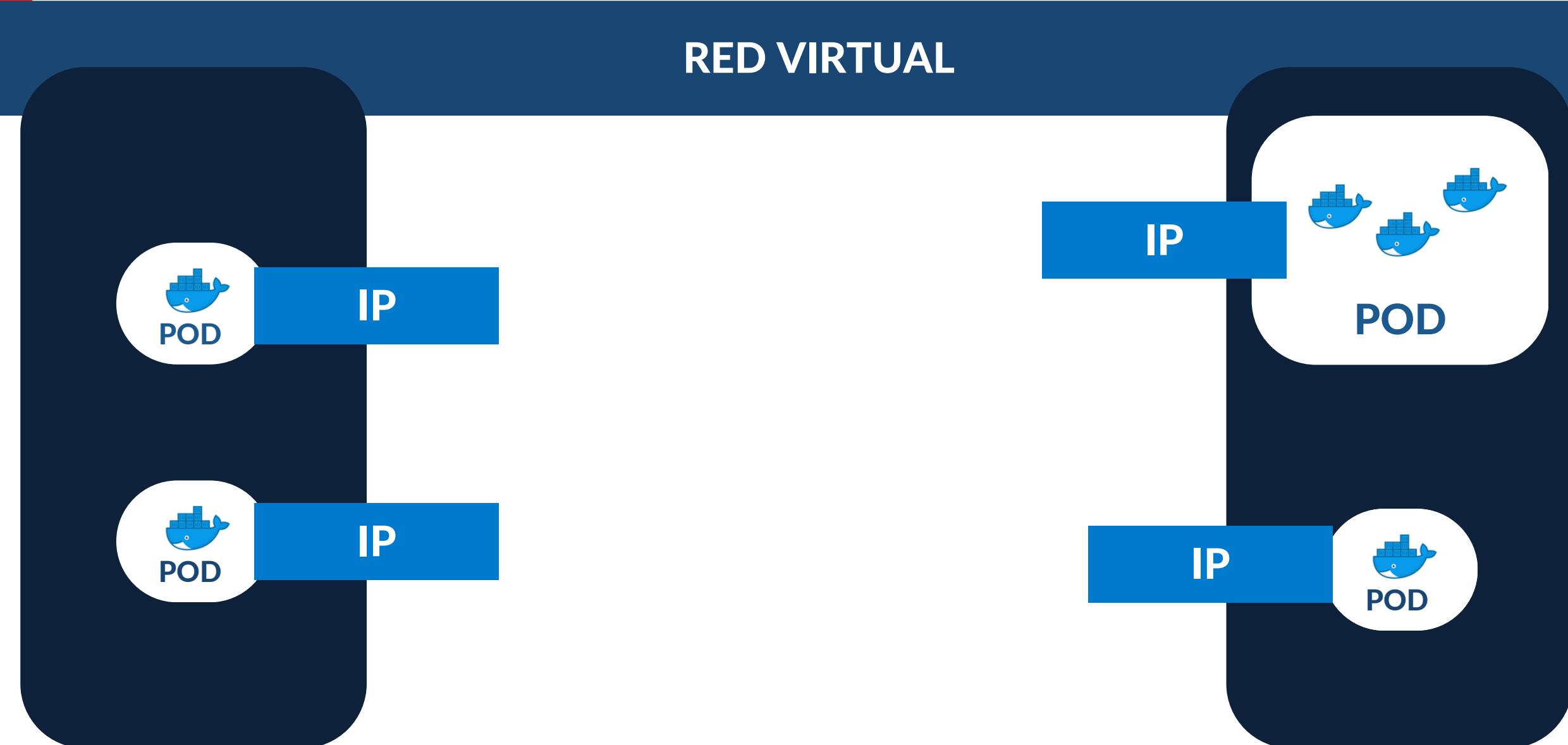
PODs



POD: Unidad más pequeña, es efímera, con su propia IP estática e ID único



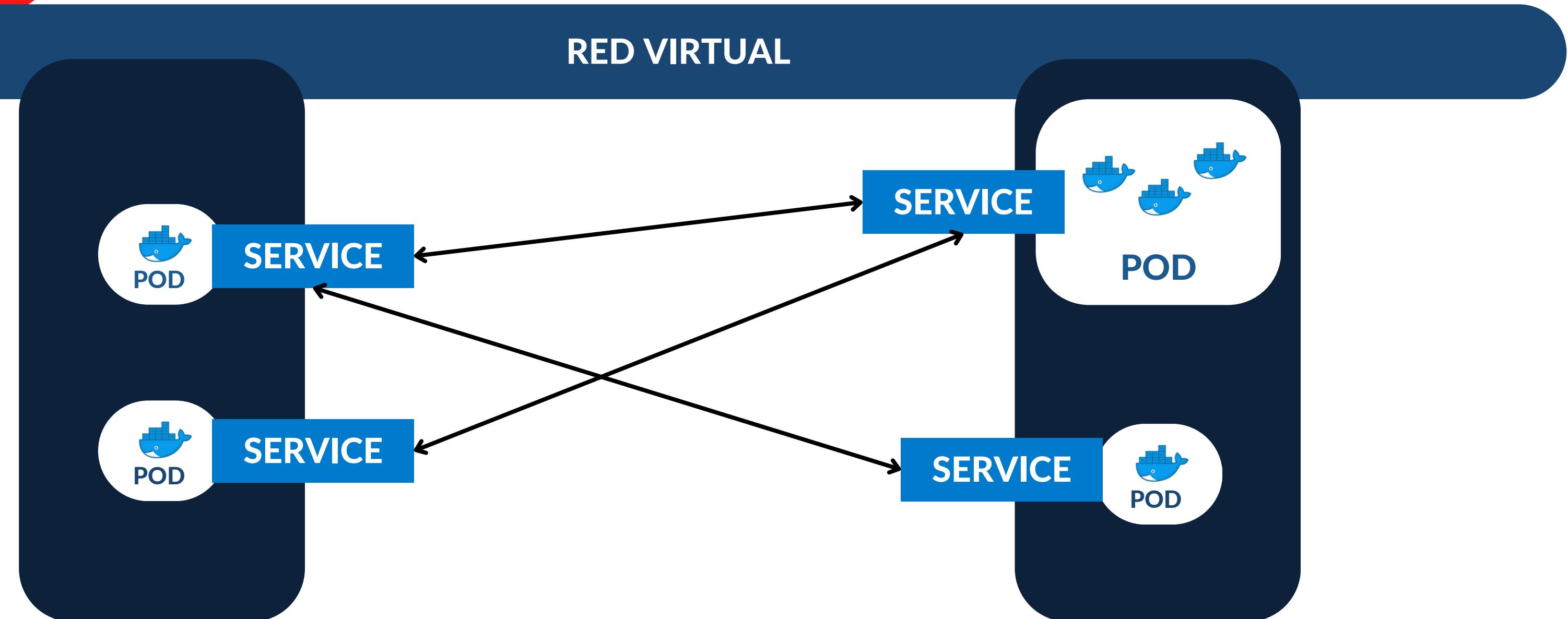
PODs



POD: Unidad más pequeña, es efímera, con su propia IP estática e ID único



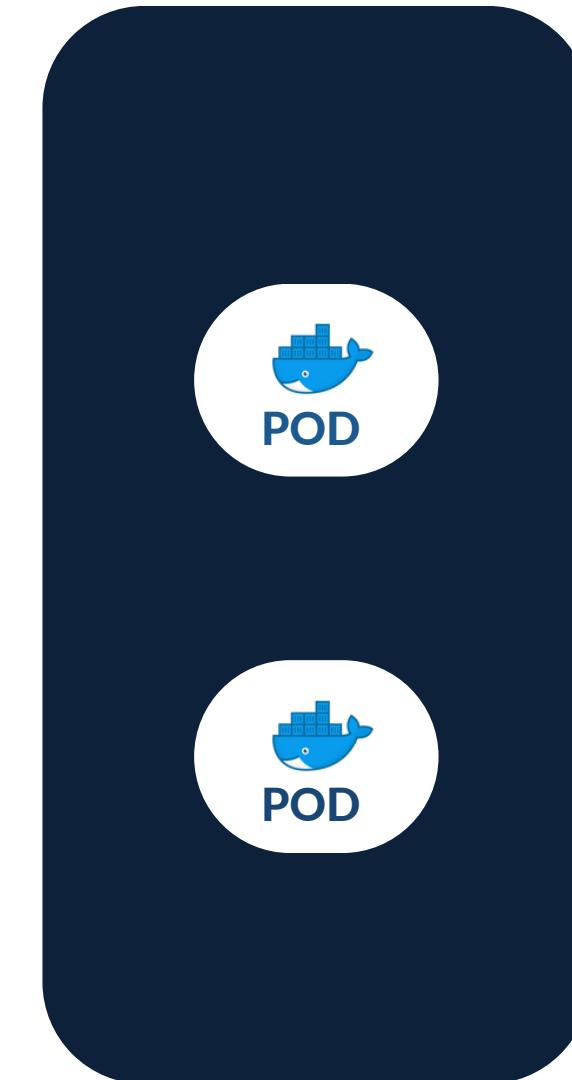
Services



Servicios: En lugar de la IP, usamos el servicio, se coloca antes del POD, y tiene su IP estática que no cambia



Secrets



Database-Secrets

- **Key=value**
- **mongo_uri=mongoXXXX**
- **postgre_uri=postgresXXX**
- **db_user=fernando**
- **db_pass=XXXXXX**

Auth-Secrets

- **jwt_secret=MiSuperSecretoJwtSeed**

Secrets: Son valores base64 que se configuran para no tener expuesto sus valores en archivos o repositorios.

