ESTILO ARQUITECTÓNICO DE MICROSERVICIOS

OBJETIVOS

- Comprender la estructura del estilo arquitectónico.
- Evaluar el uso de este estilo arquitectónico en base a sus ventajas y desventajas.
- Reconocer cuáles son los requerimientos de calidad que cubre este estilo arquitectónico.

> Muchos estilos son nombrados luego de identificar un patrón repetitivo [...]. Esto no ocurrió con los Microservicios.
>
> -- Mark Richards & Neal Ford (2020)

1

TRASFONDO

¿Cómo se concibió este estilo arquitectónico?

TRASFONDO DEL ESTILO DE MICROSERVICIOS

- Popularizado por Martin Fowler & James Lewis en Marzo 2014.
  - Definición de la Arquitectura.
  - Filosofía de la Arquitectura.
- Inspirado en DDD.
- Alto desacoplamiento.
  - Duplicación funcional sobre reuso.

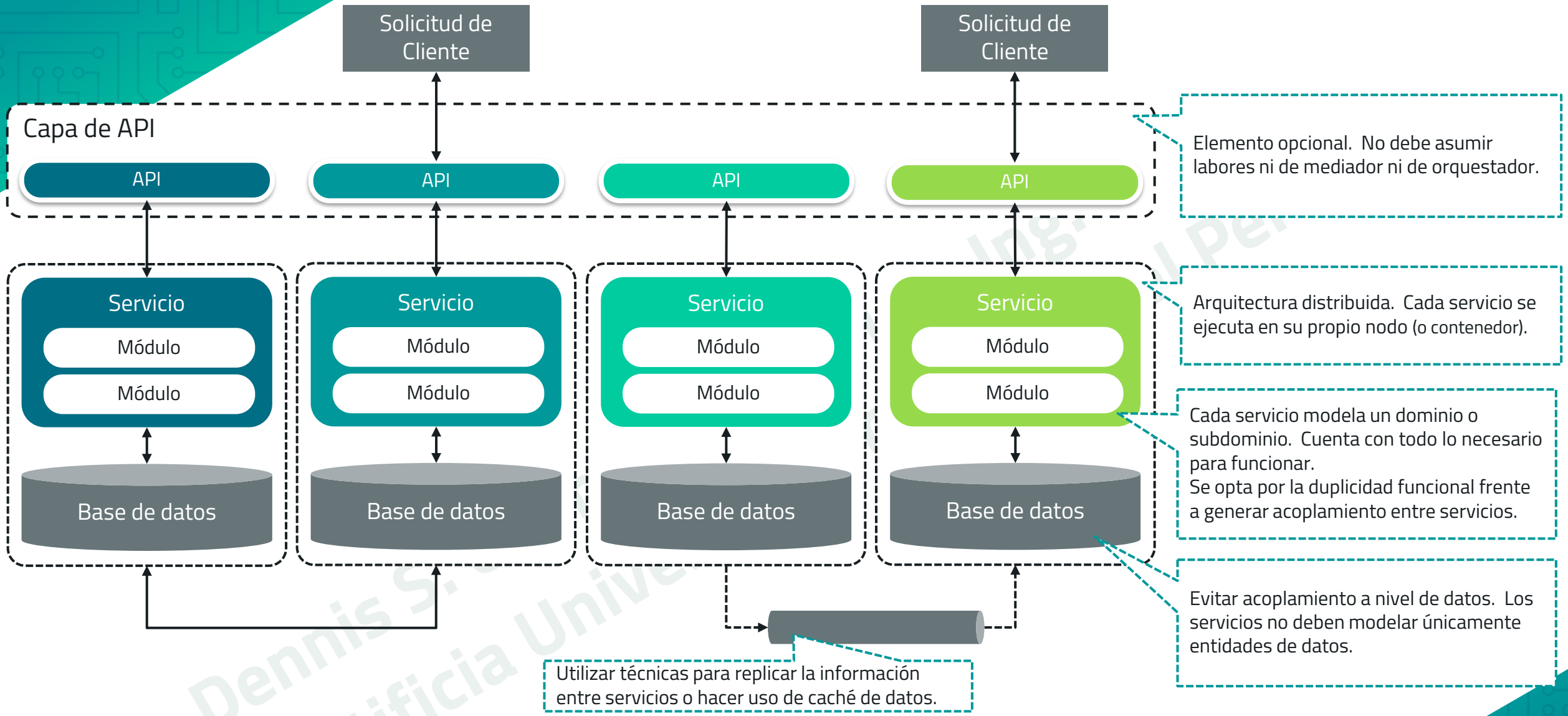
2

TOPOLOGÍA

¿Cómo se estructuran sus componentes?

TOPOLOGÍA DEL ESTILO DE MICROSERVICIOS

- Partición por dominio.
- Servicios de menor tamaño (funcional).
- Servicios independientes.



COMPONENTES COMUNES DE LA TOPOLOGÍA DE MICROSERVICIOS

GRANULARIDAD

Una transacción no debe escapar del contexto de su microservicio; es decir, no debe ser distribuida.

> Microservicios es una etiqueta;
> NO es una descripción.
>
> -- Martin Fowler

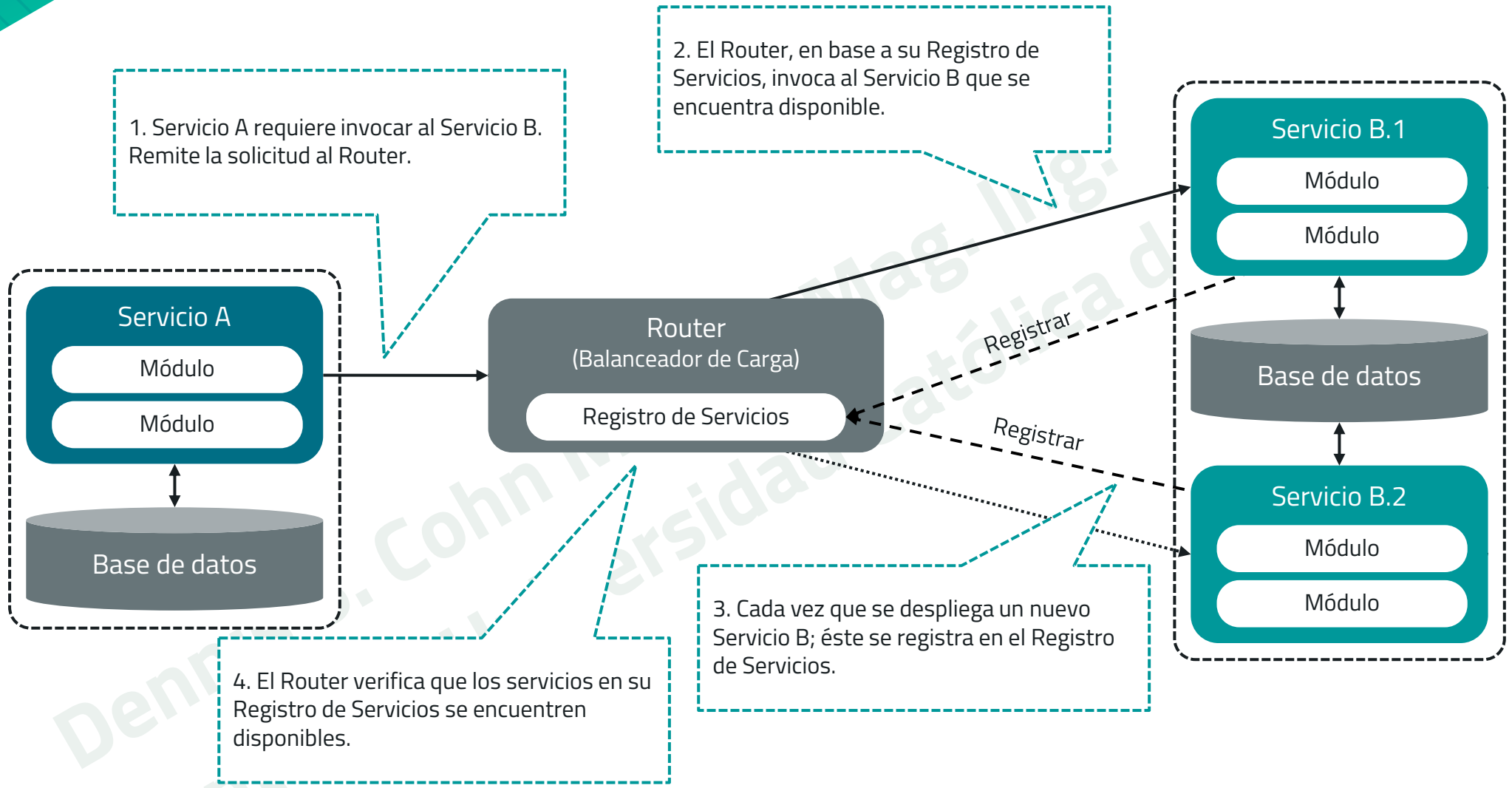
3

ESTRATEGIAS

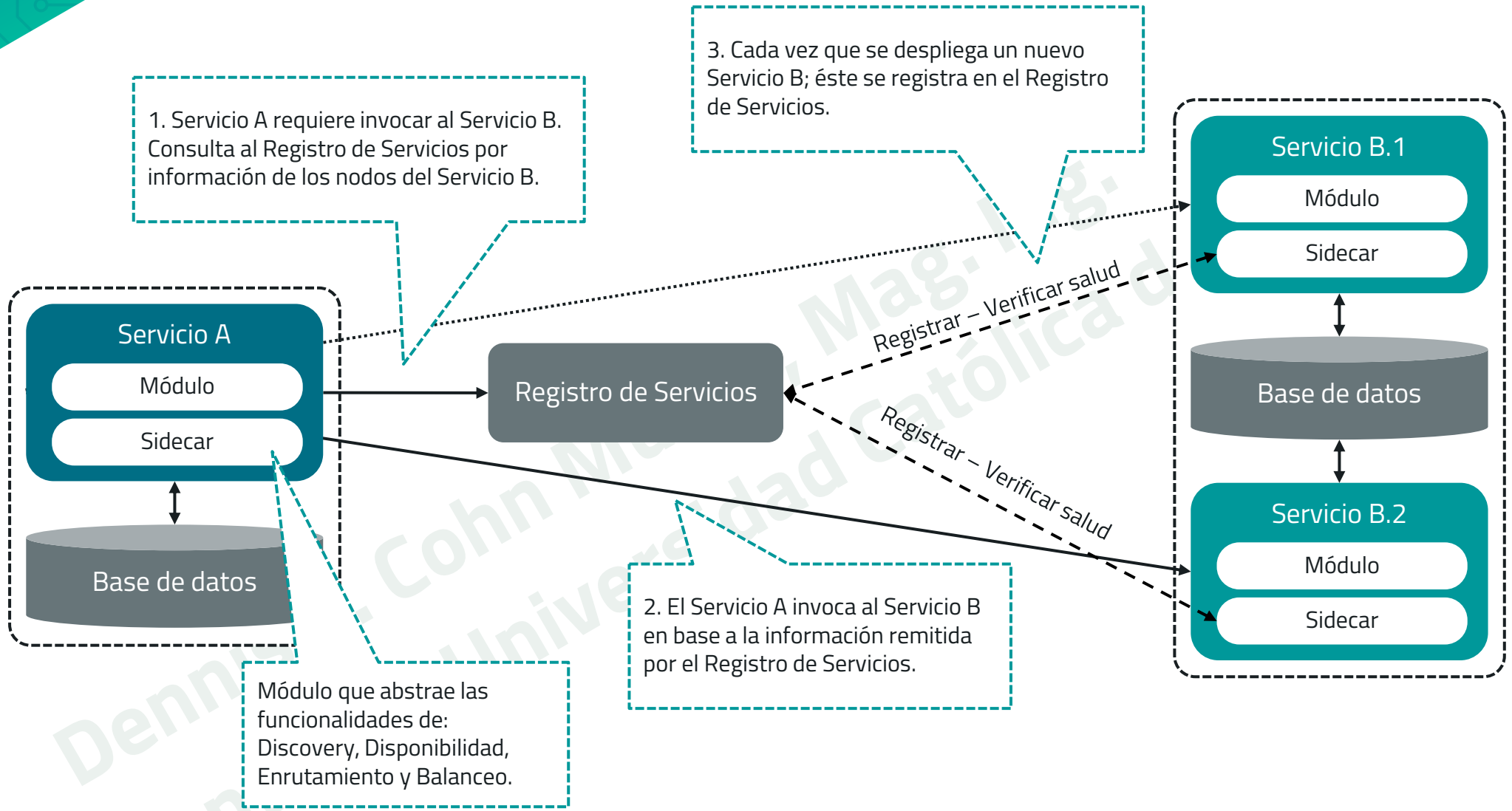
Estrategias para cubrir las limitaciones del estilo

INVOCACIÓN A MICROSERVICIOS

- Cada microservicio no conoce el estado de salud de los demás servicios.
- Cada microservicio no conoce si los servicios de los que depende se encuentran activos.
- Cada microservicio no conoce la ruta para invocar los servicios de los que depende.
  - Los Servicios son volátiles y pueden cambiar su ruta de forma dinámica.



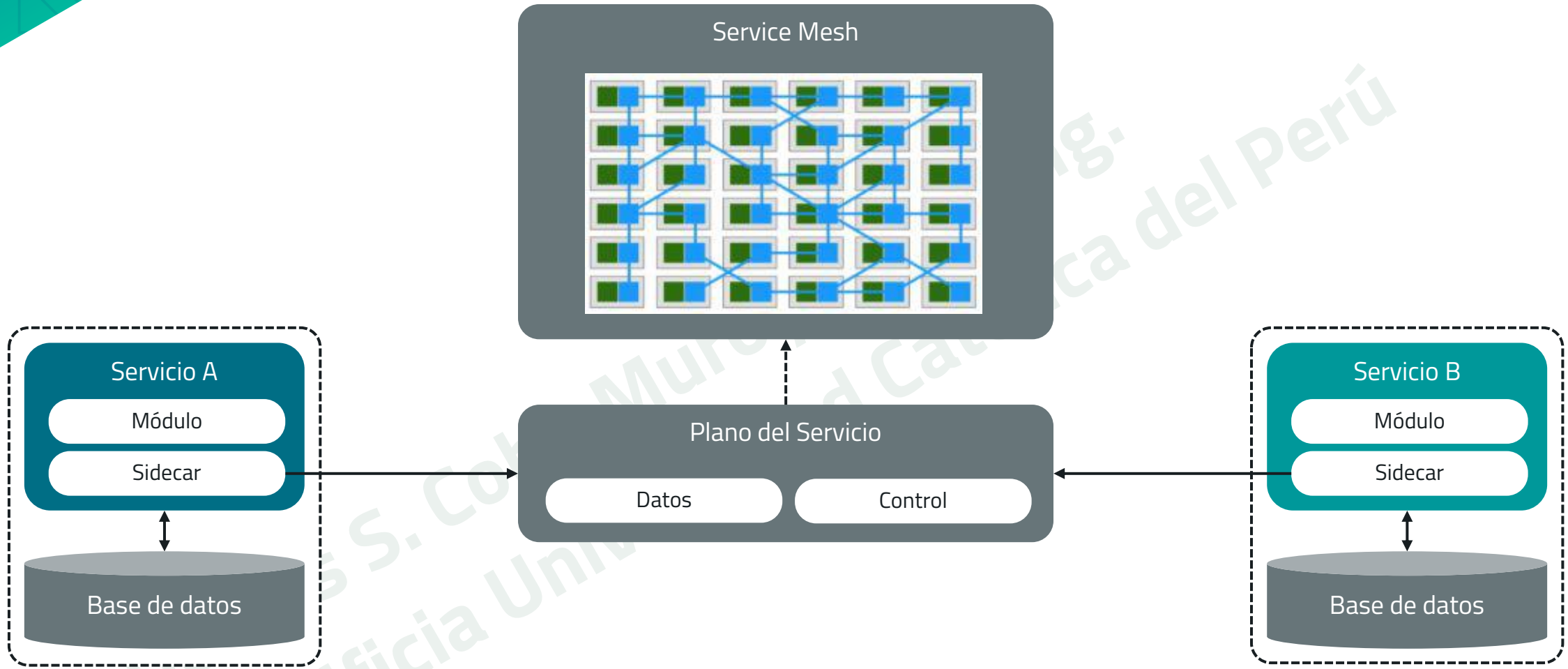
SERVICE DISCOVERY DE LADO DEL SERVIDOR



SERVICE DISCOVERY DE LADO DEL CLIENTE

FUNCIONALIDADES OPERATIVAS COMUNES

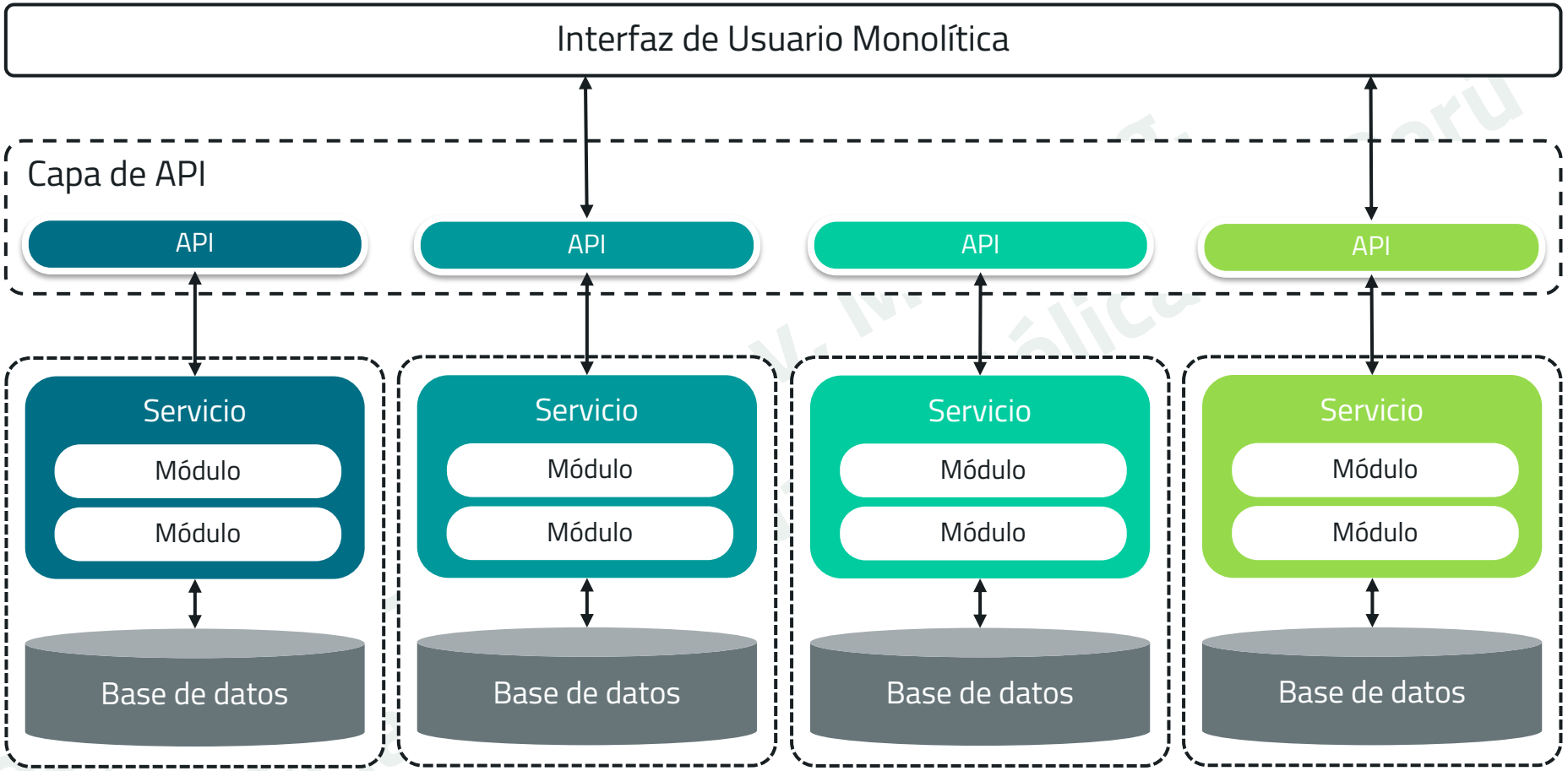
- Patrón “sidecar”
- Cada servicio incluye un “sidecar” común:
  - Descubrimiento de servicios (Service Discovery).
  - Comprobación de disponibilidad (Health check).
  - Enrutamiento y balanceo de carga: time-out, reintentos, circuit-breaker, fail-over.
  - Seguridad y control de acceso (autenticación / autorización).
  - Observabilidad: monitoreo y logs.



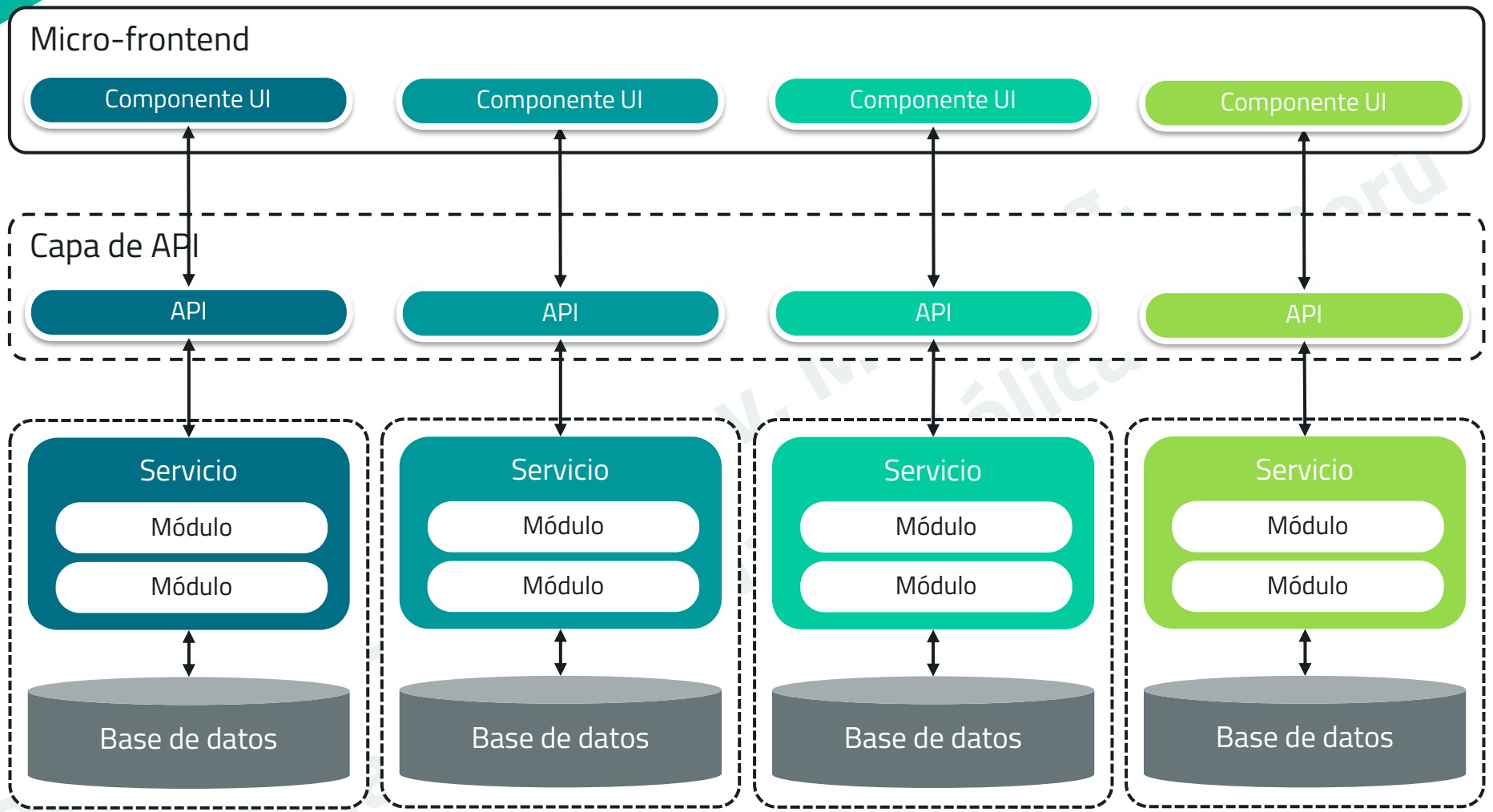
REUSO OPERACIONAL: SIDECAR

FRONTENDS

- Frontend monolítico
  - Aplicación de escritorio.
  - Aplicación móvil.
  - Aplicación web.
- Micro-frontends



FRONTEND MONOLÍTICO



MICRO-FRONTENDS

COMUNICACIÓN ENTRE SERVICIOS

- La comunicación puede ser:
  - Síncrona (REST)
  - Asíncrona (eventos-mensajes)
- Cada servicio, implementado bajo su propia tecnología, puede (y sabe cómo) invocar a otros servicios. No hace uso de un “Hub” central.

COMUNICACIÓN: COREOGRAFÍA

- Basado en el estilo arquitectónico dirigido por eventos de tipo Broker.
- Cada servicio llama a otros de forma directa.

COMUNICACIÓN: ORQUESTAMIENTO

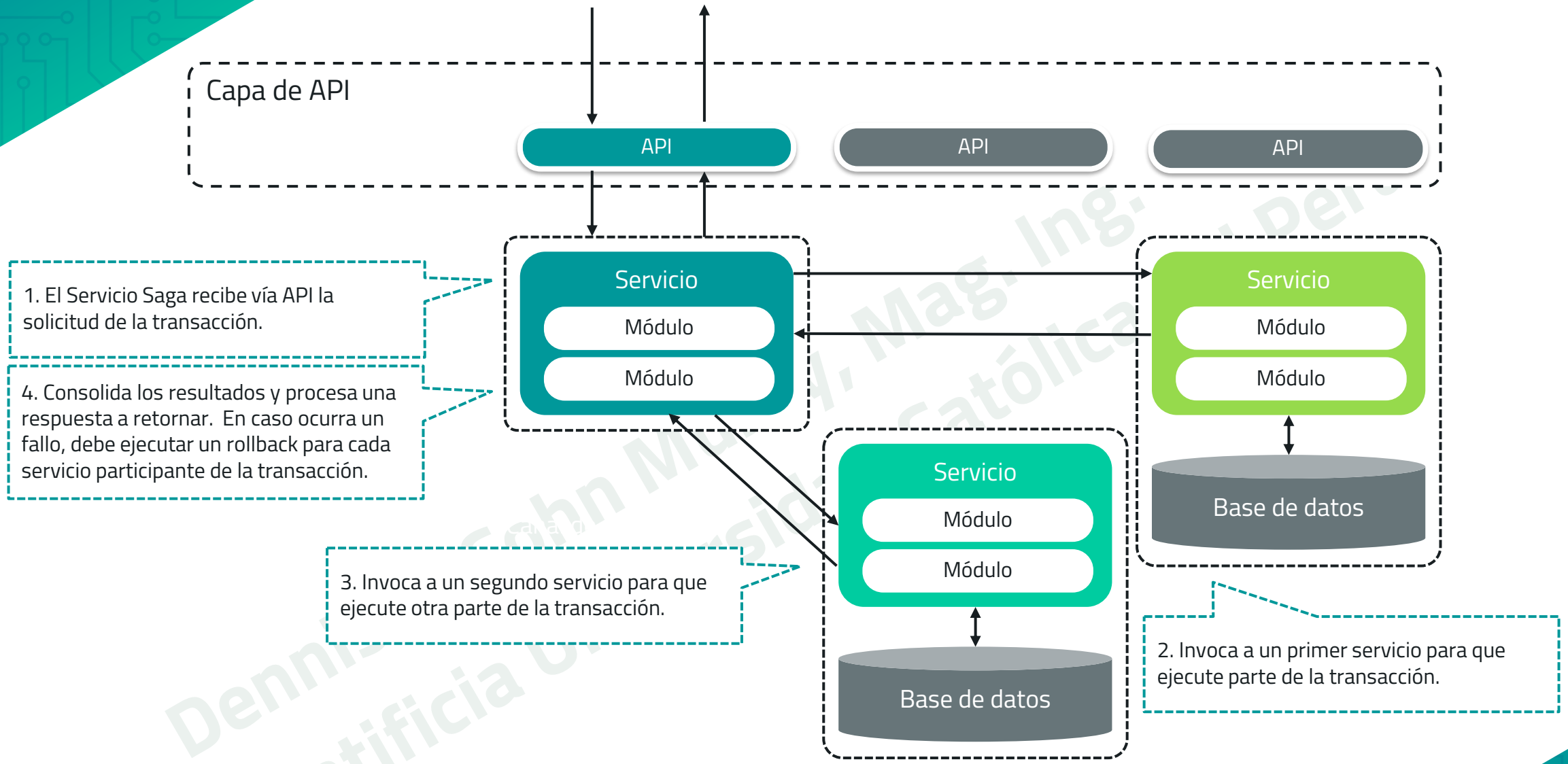
- Basado en el estilo arquitectónico dirigido por eventos de tipo Mediador.
- Hace uso de un servicio intermedio: Mediador.
- Genera acoplamiento.

TRANSACCIONES

Patrón *Saga*: Cuando se requiere ejecutar transacciones entre servicios.

COMUNICACIÓN: TRANSACCIONES & SAGAS

- Justificar el por qué se debe dividir una transacción entre varios servicios.
- Patrón “Saga”:
  - Un servicio actúa como Mediador entre servicios.
  - Registra éxitos y fallas.
  - Coordina resultados.
  - En caso de falla debe gestionar el “rollback”.


Capa de API
API
API
API
Servicio
Módulo
Módulo
Servicio
Módulo
Módulo
Base de datos
Servicio
Módulo
Módulo
Base de datos
1. El Servicio Saga recibe vía API la solicitud de la transacción.
4. Consolida los resultados y procesa una respuesta a retornar. En caso ocurra un fallo, debe ejecutar un rollback para cada servicio participante de la transacción.
3. Invoca a un segundo servicio para que ejecute otra parte de la transacción.
2. Invoca a un primer servicio para que ejecute parte de la transacción.

PATRÓN SAGA

TECNOLOGÍA DETRÁS DE MICROSERVICIOS

Contenedores

- Docker
- Kubernetes
- Containerd
- rkt (rocket)
- OpenShift
- Amazon ECS
- Microsoft Azure Kubernetes Service (AKS)

Service Discovery

- Consul (service discovery & service mesh)
- Eureka (service registry & discovery server)
- Apache Zookeeper (service discovery)
- Nacos (service discovery & service management)
- Kubernetes Service Discovery

API Gateway

- NGINX
- Kong
- AWS API Gateway
- Spring Cloud Gateway
- Azure API Management
- Tyk
- WSO2 API Manager

Messaging

- RabbitMQ
- Apache Pulsar
- AWS Simple Queue Service (SQS)
- Azure Service Bus
- Apache ActiveMQ.

3

VENTAJAS & DESVENTAJAS

CONSIDERACIÓN AL UTILIZAR ESTE ESTILO

Ventajas

- Facilita el despliegue automático.
- Reduce la complejidad en las pruebas (cada servicio cubre un contexto).
- Se requiere de DevOps para su optimización.
- Alta escalabilidad y elasticidad.

Desventajas

- Complejidad para garantizar tolerancia a fallos y fiabilidad.
  - Alta comunicación entre servicios.
  - Mitigable con redundancia.
- Problemas de desempeño debido a latencia de red y validaciones de seguridad por servicio.
  - Utilizar caché de datos.
  - Implementar replicación de información.

5

RESUMEN

Cobertura de requisitos de calidad

CUADRO RESUMEN

Arquitectura Distribuida.

Partición por Dominio.

Número de Quantas: 1 a muchos

Requisito de Calidad	Calificación
Capacidad de ser modificado	5 / 5
Costos	1 / 5
Desempeño	2 / 5
Elasticidad	5 / 5
Escalabilidad	5 / 5
Estabilidad	4 / 5
Facilidad para ser desplegado	4 / 5
Facilidad para ser probado	4 / 5
Modularidad	5 / 5
Simplicidad	1 / 5
Tolerancia a fallos	4 / 5

6

REFERENCIAS

BIBLIOGRAFÍA

- Richards, M., & Ford, N. (2020). Fundamentals of software architecture: an engineering approach. O'Reilly Media.
- Ford, N., Richards, M., Sadalage, P., & Dehghani, Z. (2021). Software Architecture: The Hard Parts. O'Reilly Media, Inc.
- Lewis, J., & Fowler, M. (2014). Microservices: a definition of this new architectural term. URL: http://martinfowler. com/articles/microservices. html

Créditos:

- Plantilla de la presentación por SlidesCarnival
- Diseño del fondo Hero Patterns