Módulo 5: Arquitectura y escalabilidad

Arquitecturas monolíticas y microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

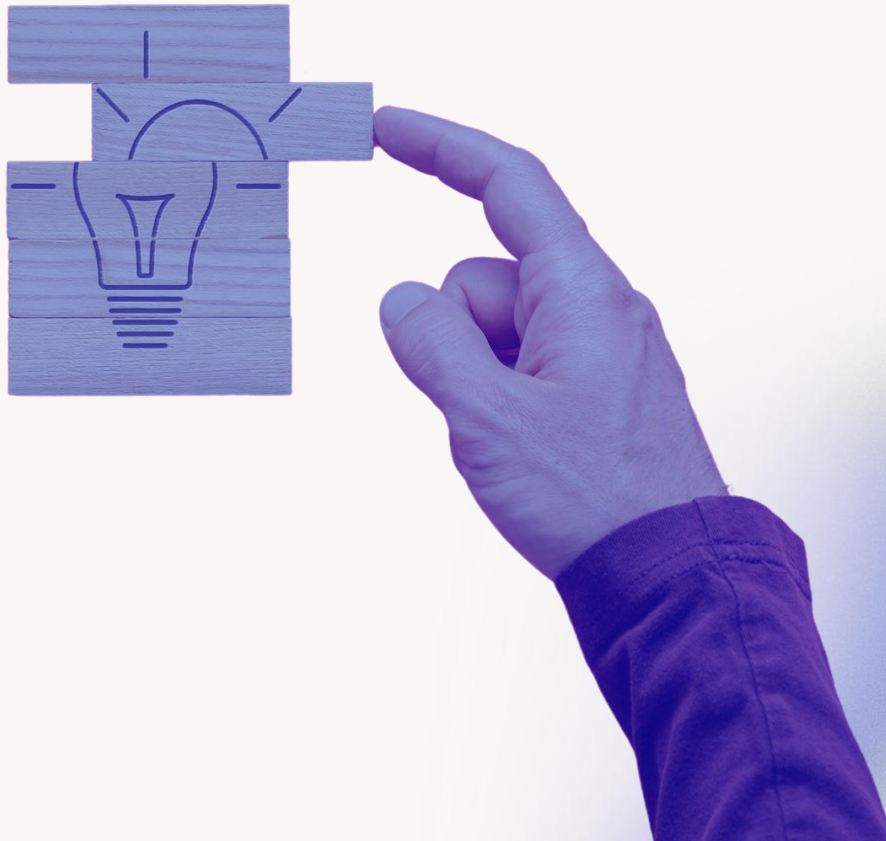
MÓDULO 5

- Principios fundamentales de diseño de una arquitectura
- ***Arquitecturas monolíticas y microservicios***
- Fundamentos de tecnologías de contenedores
- Orquestación de contenedores con Kubernetes
- Registro de contenedores
- Platform as a Service, Backend as a Service y Frontend as a Service

aspasia
LA FORMACIÓN DE TU FUTURO

Objetivos

Analizar la evolución de las arquitecturas de software desde modelos monolíticos hacia microservicios, y cómo DevOps potencia la resiliencia, escalabilidad y automatización en estos entornos.



aspasia
LA FORMACIÓN DE TU FUTURO

¿Qué cambios arquitectónicos suelen ser necesarios cuando una organización adopta prácticas DevOps?

aspasia
LA FORMACIÓN DE TU FUTURO

Cómo DevOps afecta la arquitectura

Relación entre DevOps y la arquitectura de software

aspasia
LA FORMACIÓN DE TU FUTURO

📌 La arquitectura de software es la base sobre la cual se construyen y despliegan las aplicaciones. **DevOps influye directamente en cómo se diseña esa arquitectura**, ya que fomenta:

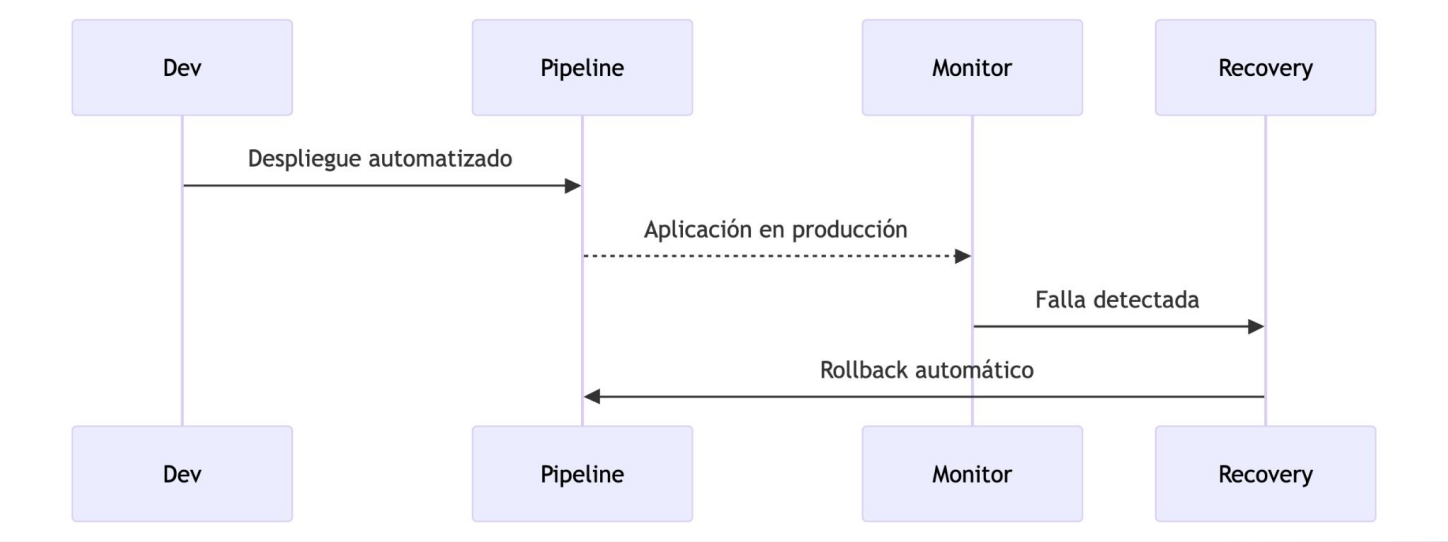
✅ **Integración continua, entrega continua y automatización**.
✅ **Modularidad y desacoplamiento** para permitir despliegues independientes.
✅ **Infraestructura como código**, necesaria para entornos reproducibles.
✅ **Feedback rápido y validación temprana** en el ciclo de desarrollo.

📖 **Ejemplo de influencia directa:**
Un sistema diseñado para DevOps evitará grandes componentes monolíticos difíciles de probar, optando por unidades pequeñas y desplegables de forma independiente.

Impacto de DevOps en la resiliencia de sistemas

aspasia
LA FORMACIÓN DE TU FUTURO

📌 La **resiliencia** en DevOps implica diseñar arquitecturas que soporten fallos y se recuperen automáticamente.



Impacto de DevOps en la resiliencia de sistemas

aspasia
LA FORMACIÓN DE TU FUTURO

✅ **Prácticas clave de resiliencia influenciadas por DevOps:**

- **Redundancia automática** mediante orquestadores como Kubernetes.
- **Autoescalado dinámico** ante picos de carga.
- **Rollback automatizados** ante fallos de despliegue.
- **Observabilidad continua** con herramientas como Prometheus, Grafana, o ELK.

Automatización y arquitectura adaptable en entornos DevOps

aspasia
LA FORMACIÓN DE TU FUTURO

📌 DevOps requiere arquitecturas **adaptables y automáticas**, capaces de evolucionar constantemente.

✅ **Características de una arquitectura adaptable:**

- Configuraciones externalizadas.
- Microservicios desplegables por separado.
- Infraestructura gestionada con herramientas como Terraform o Ansible.

📖 **Ejemplo de definición adaptable con Docker Compose:**

```
services:
  api:
    build: .
    environment:
      - CONFIG_PATH=/config/api
    volumes:
      - ./config:/config
```

Integración de CI/CD en arquitecturas modernas

aspasia
LA FORMACIÓN DE TU FUTURO

📌 Una arquitectura moderna debe facilitar la automatización completa del flujo de desarrollo: **desde el commit hasta producción**.

✅ **Elementos arquitectónicos que soportan CI/CD:**

- Versionamiento de configuraciones.
- Contenedores con dependencias controladas.
- Separación clara entre build y runtime.

📖 **Ejemplo de pipeline básico con GitHub Actions para microservicio:**

```yaml
name: Build and Test

on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm install
      - run: npm test
```

Seguridad y gobernanza en arquitecturas DevOps

aspasia
LA FORMACIÓN DE TU FUTURO

📌 DevOps impulsa una cultura de **seguridad desde el inicio**, también conocida como **DevSecOps**.

✅ **Prácticas de seguridad integradas:**

- Análisis de dependencias con SCA (Software Composition Analysis).
- Escaneo de contenedores en CI (e.g., Trivy).
- Gestión de secretos (e.g., Vault, GitHub Secrets).
- Políticas de acceso mínimas en infraestructura.

📖 **Ejemplo – GitHub Action para escaneo de dependencias:**

```
- name: Run Trivy vulnerability scanner
  uses: aquasecurity/trivy-action@v0.11.2
  with:
    image-ref: my-app:latest
```

DevOps como facilitador de la arquitectura escalable

aspasia
LA FORMACIÓN DE TU FUTURO

📌 Una arquitectura bien diseñada para DevOps **soporta crecimiento**, tanto en tráfico como en equipos de desarrollo.

✅ **Cómo DevOps lo hace posible:**

- Orquestación con Kubernetes para escalar horizontalmente.
- Desacoplamiento de servicios para escalar por componentes.
- Observabilidad y monitoreo para detectar cuellos de botella.

📖 **Ejemplo de autoscaling en Kubernetes:**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler

spec:
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 70
```

Reducción del time-to-market con arquitecturas optimizadas para DevOps

aspasia
LA FORMACIÓN DE TU FUTURO

📌 DevOps acorta significativamente el tiempo entre la idea y el producto en producción.

✅ **Factores que contribuyen a un time-to-market bajo:**

- Automatización de pruebas, build y despliegue.
- Feedback temprano en el ciclo de vida.
- Arquitecturas modulares que evitan bloqueos entre equipos.

📖 **Ejemplo real:**

En lugar de esperar una entrega monolítica trimestral, un equipo con arquitectura DevOps puede entregar nuevas features **cada semana o incluso varias veces al día.**

Ejemplos de éxito en la implementación de DevOps en la arquitectura

aspasia
LA FORMACIÓN DE TU FUTURO

✅ Netflix

- Arquitectura basada en microservicios.
- CI/CD completamente automatizado.
- Pruebas de resiliencia con herramientas como Chaos Monkey.

✅ Amazon

- Infraestructura completamente desacoplada.
- Despliegues independientes por equipo/producto.
- Monitorización, autoscaling y rollback automatizado.

aspasia
LA FORMACIÓN DE TU FUTURO

¿Qué beneficios concretos aporta DevOps a la resiliencia, escalabilidad y velocidad de entrega en una arquitectura?

aspasia
LA FORMACIÓN DE TU FUTURO

Arquitecturas monolíticas

Definición y características de una arquitectura monolítica

aspasia
LA FORMACIÓN DE TU FUTURO

📌 Una **arquitectura monolítica** es aquella donde todos los componentes de una aplicación (UI, lógica de negocio, acceso a datos) están agrupados en una única unidad de despliegue.

✅ **Características principales:**

- Código fuente en un solo repositorio o módulo.
- Despliegue unificado.
- Compartición directa de memoria y clases entre módulos.
- Acoplamiento fuerte entre componentes.

Ventajas y desventajas del enfoque monolítico

aspasia
LA FORMACIÓN DE TU FUTURO

❌ **Desventajas:**

- Difícil de escalar por componente.
- Alto acoplamiento impide entregas independientes.
- Cambios en una parte pueden afectar toda la aplicación.
- Complejidad creciente con el tamaño del sistema.

✅ **Ventajas:**

- Fácil de desarrollar en etapas tempranas.
- Despliegue y testing simplificados.
- Buen rendimiento para sistemas pequeños/medianos.

📖 **Ejemplo:**

Una app eCommerce donde el sistema de pagos, carrito y usuarios comparten el mismo código base y se despliegan juntos.

Desafíos en la evolución de sistemas monolíticos

aspasia
LA FORMACIÓN DE TU FUTURO

❌ A medida que el sistema crece, los siguientes problemas se intensifican:

- **Riesgo elevado en despliegues** (todo el sistema cae ante un fallo).
- **Frenos en el desarrollo paralelo** (bloqueos por cambios compartidos).
- **Dificultades de mantenimiento** por lógica enredada.
- **Adopción de nuevas tecnologías** limitada por la dependencia común.

📖 **Ejemplo:**
Un cambio en la lógica de login puede requerir pruebas y despliegue de todo el sistema, afectando incluso funcionalidades no relacionadas.

Patrones de optimización para arquitecturas monolíticas

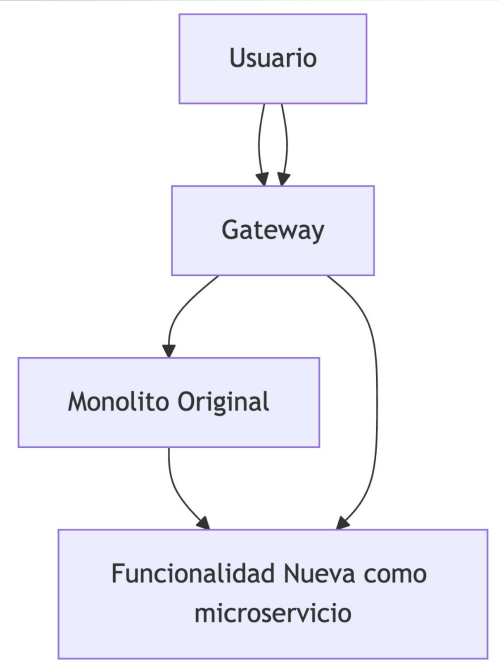
aspasia
LA FORMACIÓN DE TU FUTURO

✅ Antes de migrar, los monolitos pueden ser optimizados aplicando ciertos patrones:

- **Modularización interna:** dividir el código en paquetes o módulos bien definidos.
- **Separación de capas:** UI, lógica y persistencia con responsabilidades claras.
- **Uso de gateways o wrappers** para preparar servicios desacoplables.
- **Base de datos compartida pero con acceso aislado.**

📖 **Ejemplo:**
Separar la lógica de usuario y de pagos en paquetes distintos dentro del mismo monolito, aunque compartan despliegue.

Estrategias para la transición de monolitos a microservicios



📌 Esta transición debe ser gradual y planificada. Algunas estrategias incluyen:

✅ **Enfoques comunes:**

- **Strangler pattern:** nuevos módulos se desarrollan como servicios externos, "estrangulando" el monolito.
- **Identificación de límites de dominio (DDD).**
- **Desacoplamiento por capas o funcionalidades.**
- **Externalización de una funcionalidad crítica (e.g., login, notificaciones).**

Impacto de la modularización en sistemas monolíticos

aspasia
LA FORMACIÓN DE TU FUTURO

📌 La **modularización** permite un mejor mantenimiento, pruebas independientes, y sirve como **puente hacia microservicios**.

✅ **Beneficios:**

- Menor acoplamiento interno.
- Reducción de bugs por colisión de cambios.
- Mayor facilidad para hacer refactorización progresiva.
- Aumento de la reusabilidad de módulos.

📖 **Ejemplo:**

En un monolito modular, la lógica de pagos podría evolucionar sin afectar directamente al sistema de facturación o stock.

Ejemplos y casos de uso en la industria

aspasia
LA FORMACIÓN DE TU FUTURO

- **WordPress (PHP):**
  - Arquitectura monolítica con gran éxito.
  - Plugins y plantillas permiten cierta modularidad.
  - Escala a través de cachés y servidores replicados.
- **ERP tradicionales (SAP, Oracle):**
  - Inicialmente monolíticos.
  - Han evolucionado hacia arquitecturas híbridas con servicios web.
- **GitLab:**
  - Comenzó como monolito Ruby on Rails.
  - Implementó modularización progresiva y ahora contiene microservicios para tareas como CI/CD y runners.

aspasia
LA FORMACIÓN DE TU FUTURO

¿Qué retos enfrentan las organizaciones al migrar de un monolito a una basada en microservicios?

aspasia
LA FORMACIÓN DE TU FUTURO

Arquitecturas basadas en microservicios

Definición y características de una arquitectura basada en microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

📌 La arquitectura de microservicios es un estilo que estructura una aplicación como un conjunto de servicios pequeños, independientes, que se comunican entre sí a través de APIs.

✅ **Características principales:**

- Cada servicio se enfoca en una única funcionalidad del negocio.
- Servicios desplegables y escalables de forma independiente.
- Comunicación generalmente basada en HTTP/REST, gRPC o mensajería.
- Tecnología agnóstica (cada servicio puede usar su lenguaje o base de datos).

Comparación entre arquitecturas monolíticas y microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

Aspecto	Monolítico	Microservicios
Despliegue	Único y conjunto	Independiente por servicio
Escalabilidad	Global (escala todo)	Por componente
Tecnología	Homogénea	Heterogénea
Tiempo de desarrollo	Rápido al inicio	Más complejo inicialmente
Aislamiento de fallos	Bajo	Alto (cada servicio puede aislar errores)
Curva de aprendizaje	Baja	Alta (requiere más conceptos y herramientas)

Ventajas y desventajas de la adopción de microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

✅ **Ventajas:**

- Despliegues independientes.
- Escalado horizontal específico.
- Mejor resiliencia del sistema.
- Favorece equipos autónomos y delivery rápido.

❌ **Desventajas:**

- Mayor complejidad operativa.
- Requiere cultura DevOps y automatización.
- Necesidad de manejar fallos distribuidos.
- Observabilidad más compleja.

📖 **Ejemplo de mejora:**
Migrar el sistema de login de un monolito a un microservicio permite actualizar o escalar autenticación sin tocar el resto del sistema.

Mecanismos de escalamiento en arquitecturas de microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

📌 Cada servicio se escala individualmente en función de su carga o demanda.

✅ **Estrategias comunes:**

- Kubernetes Horizontal Pod Autoscaler.
- Serverless (e.g., AWS Lambda para funciones específicas).
- Partitioning por dominio (sharding de datos).

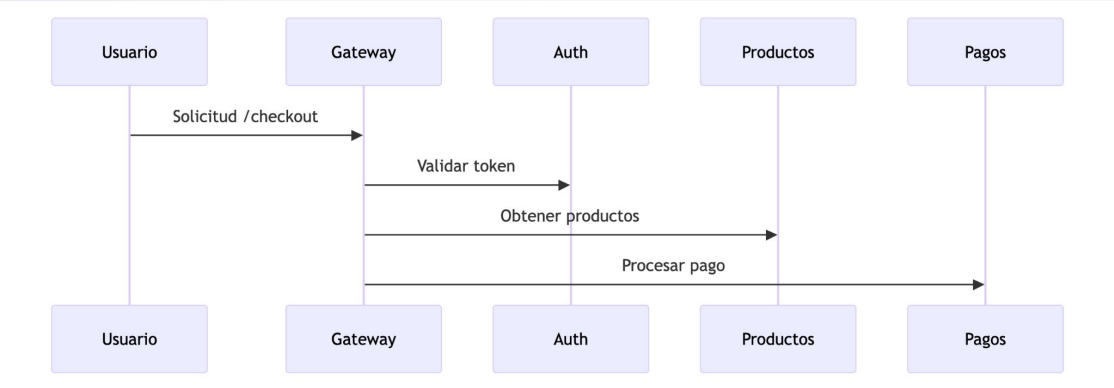
📖 **Ejemplo YAML de autoscaling:**

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler

metadata:
  name: payment-service-hpa

spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: payment-service
  minReplicas: 2
  maxReplicas: 10

  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Rol del proxy y el gateway en arquitecturas de microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

📌 En una arquitectura distribuida, el **API Gateway** es el punto de entrada que abstrae la complejidad interna del sistema.



✅ **Funciones del Gateway:**

- Ruteo de peticiones a los servicios correctos.
- Autenticación y autorización centralizada.
- Límite de tasa (rate limiting).
- Registro y métricas.

Estrategias de comunicación y orquestación de microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

📌 Los microservicios se comunican entre sí usando diferentes patrones, según los casos:

✅ **Patrones de comunicación:**

- **Síncrona (REST, gRPC):** útil para flujos que requieren respuesta inmediata.
- **Asíncrona (eventos, colas de mensajes):** ideal para desacoplar procesos (RabbitMQ, Kafka).

✅ **Orquestación vs Coreografía:**

- **Orquestación:** un servicio central dirige las interacciones.
- **Coreografía:** los servicios reaccionan a eventos sin un coordinador central.

Buenas prácticas para la implementación de microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

✅ **Sugerencias clave:**

- Diseña servicios alrededor de capacidades del negocio (Bounded Context).
- Automatiza testing, despliegue y monitoreo desde el inicio.
- Implementa circuit breakers y retry para tolerancia a fallos.
- Versiona APIs y evita contratos rotos entre servicios.

📖 **Herramientas recomendadas:**

- Observabilidad: **Prometheus + Grafana**, **OpenTelemetry**
- API Gateway: **Kong**, **NGINX**, **Istio**
- CI/CD: **Jenkins**, **GitHub Actions**, **ArgoCD**

Casos de éxito y adopción de microservicios en la industria

aspasia
LA FORMACIÓN DE TU FUTURO

- **Netflix**
  - Pioneer del enfoque de microservicios.
  - Cada microservicio puede ser desarrollado y desplegado por un equipo independiente.
  - Observabilidad y resiliencia a través de Chaos Engineering.
- **Spotify**
  - Equipos autónomos llamados “squads”.
  - Microservicios separados por funcionalidades: recomendaciones, reproducción, cuentas.
- **Uber**
  - Más de 1000 microservicios en producción.
  - Uso intensivo de orquestación, tracing y balanceo global.

aspasia
LA FORMACIÓN DE TU FUTURO

¿Cómo contribuyen los microservicios a la escalabilidad y flexibilidad de las aplicaciones en entornos DevOps?

aspasia
LA FORMACIÓN DE TU FUTURO

Ejercicio Guiado: Transición de Arquitectura Monolítica a Microservicios en un Entorno DevOps

aspasia
LA FORMACIÓN DE TU FUTURO

Transición de Arquitectura Monolítica a Microservicios

En este ejercicio, simularás que formas parte del equipo técnico de una empresa que ofrece una **plataforma digital para arrendamiento y venta de inmuebles**. Esta aplicación fue diseñada inicialmente como un **monolito**, pero el crecimiento de usuarios, la demanda de nuevas funcionalidades y la necesidad de escalar por zonas geográficas requiere una evolución hacia una arquitectura de **microservicios**.

aspasia
LA FORMACIÓN DE TU FUTURO

Transición de Arquitectura Monolítica a Microservicios

Tu equipo deberá **analizar, representar y justificar** esta evolución arquitectónica, considerando la integración con prácticas DevOps como CI/CD, automatización, resiliencia, y escalabilidad.

aspasia
LA FORMACIÓN DE TU FUTURO

Objetivos

- Identificar y representar una arquitectura monolítica inicial.
- Detectar los problemas asociados al escalado y mantenimiento.
- Diseñar una arquitectura basada en microservicios.
- Aplicar principios de DevOps en ambas arquitecturas.
- Comparar y justificar la transición.

🧱 Paso 1: Analizar la arquitectura monolítica actual

aspasia
LA FORMACIÓN DE TU FUTURO

📌 ¿Qué haremos?
Describiremos los módulos funcionales de la aplicación actual en su estructura monolítica.

Instrucciones:

1. En equipo, imaginen que la aplicación inmobiliaria contiene al menos estos módulos:
   - Gestión de usuarios
   - Listado de propiedades
   - Agenda de visitas
   - Procesos de pago
   - Administración de contratos
2. Dibuja cómo estos componentes están unidos dentro de una única **aplicación monolítica**.
   - Incluye base de datos única, servidor único y dependencias directas entre módulos.

📉 Paso 2: Identificar problemas del monolito

aspasia
LA FORMACIÓN DE TU FUTURO

📌 **¿Qué haremos?**
Analizaremos las limitaciones actuales y cómo afectan al negocio.

Preguntas guía:

- ¿Qué sucede si quieres escalar solo el módulo de pagos?
- ¿Qué ocurre si un error en la agenda rompe toda la aplicación?
- ¿Cómo afecta al time-to-market hacer una nueva entrega del sistema?

✅ Haz una lista de al menos **3 desafíos técnicos y de negocio**.

🧩 Paso 3: Rediseñar la arquitectura con microservicios

aspasia
LA FORMACIÓN DE TU FUTURO

📌 ¿Qué haremos?
Dividiremos los componentes en servicios independientes.

Instrucciones:

1. Usa el mismo conjunto de módulos (usuarios, propiedades, agenda, pagos, contratos).
2. Separa cada uno como **microservicio** independiente, con su propia base de datos o almacenamiento si aplica.
3. Conecta los servicios mediante una **API Gateway**.

✅ Incluye mecanismos de comunicación como REST, gRPC o eventos.

⚙️ Paso 4: Aplicar DevOps en ambas arquitecturas

aspasia
LA FORMACIÓN DE TU FUTURO

📌 ¿Qué haremos?

Incorporaremos las prácticas DevOps en los dos enfoques.

- **Monolito:**
  - CI/CD único para toda la app.
  - Despliegue manual o automatizado.
  - Requiere reinicio completo para cambios.
- **Microservicios:**
  - Pipelines independientes por servicio.
  - Automatización y pruebas por componente.
  - Despliegue continuo y segmentado.

✅ Representa cómo DevOps mejora la entrega, resiliencia y escalabilidad.

📈 Paso 5: Pensar en escalabilidad y resiliencia

aspasia
LA FORMACIÓN DE TU FUTURO

📌 **¿Qué haremos?**
Compararemos cómo escalar y mantener cada arquitectura.

Indicaciones:

- En el monolito: solo puedes escalar verticalmente (más RAM, CPU).
- En microservicios: escalas horizontalmente cada servicio por separado.
- Considera el uso de contenedores y orquestación (Kubernetes, Docker).

✅ Añade al diseño elementos de **resiliencia** (retry, circuit breakers, logging centralizado).

🧑‍🎨 Paso 6: Representar ambas arquitecturas

aspasia
LA FORMACIÓN DE TU FUTURO

📌 **¿Qué haremos?**
Dibujaremos los dos modelos para análisis comparativo.

Herramientas sugeridas:

- Draw.io
- Miro
- Lucidchart
- Papel y foto

✅ Cada equipo debe presentar un **diagrama de arquitectura monolítica** y uno de **microservicios con DevOps integrado**.

Preguntas finales

aspasia
LA FORMACIÓN DE TU FUTURO

- ¿Qué beneficios aporta la arquitectura de microservicios en este caso?
- ¿Qué retos operativos y técnicos surgirían con esta transición?
- ¿Cómo impacta DevOps en la resiliencia y despliegue del sistema?
- ¿Qué criterios usarías para decidir si un sistema debe evolucionar o no?

Entregable:

aspasia
LA FORMACIÓN DE TU FUTURO

Cada equipo deberá entregar:

- Dos diagramas: arquitectura monolítica y microservicios.
- Una tabla comparativa de pros y contras.
- Una breve explicación de cómo se aplica DevOps en cada caso.
- (Opcional) Una propuesta de estrategia de migración progresiva por módulos.

aspasia
LA FORMACIÓN DE TU FUTURO

Resumen de lo aprendido

- **Relación entre DevOps y Arquitectura:** DevOps promueve arquitecturas más resilientes, automatizadas y escalables, integrando CI/CD.
- **Arquitectura Monolítica:** Modelo tradicional con estructura unificada; ventajas como simplicidad y desventajas como dificultad para escalar.
- **Arquitectura de Microservicios:** Enfoque distribuido que permite despliegue independiente, escalamiento granular y mayor flexibilidad en el desarrollo.
- **Transición y Buenas Prácticas:** Estrategias de modularización, uso de gateways, comunicación entre servicios y ejemplos reales de éxito.

aspasia
LA FORMACIÓN DE TU FUTURO

Próxima clase...

Fundamentos de tecnologías de contenedores


dalid
aspasia
LA FORMACIÓN DE TU FUTURO