1. Identificar los cuellos de botella en una aplicación existente (métricas, herramientas o logs que usarías).

Suponiendo que un micoservicio tuviese problemas relacionados con la latencia en una API, analizaría los logs con datadog, el uso de la cpu con clinic.js o py-spy y verificaría si hay bloqueos I/O.

Para entornos en la nube utilizo CloudWatch o GC Monitoring para analizar el rendimiento en tiempo real de los microservicios que utilizo. En el caso de servicios de logging estructurado ELK Stack o datadog

2. Planear y ejecutar la refactorización (cómo organizarías el código, separarías responsabilidades o mejorarías la arquitectura).

Una vez identificado el cuello de botella se debe planificar la refactorización enfocados en código limpio, desacoplado y óptimo.

Como estrategía me baso en los principios de clean architecture y SOLID separando lógica de negocio en servicios, reducir los monolitos dentro de los microservicios y usar patrones de diseño tradicionales.

3. Medir la mejora obtenida (qué indicadores y mediciones antes y después implementarías para evidenciar los cambios).

Después de la refactorización es clave medir resultados y validar que la optimización sea exitosa. Para realizar estos análisis, se analizan nuevamente indicadores claves de desempeño como reducciones en la latencia, uso óptimo de cpu y memoria, reducción de errores 5XX y finalmente pruebas de carga con k6 o Artillery.

Indicador	Antes	Después	Mejora
Latencia P95	800ms	120ms	85% menos
Uso de RAM	800MB	250MB	69% menos
Errores 5XX	20%	1%	Estabilidad
Requests por seg	100	400	4X más capacidad