IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 12

Arquitectura de sistemas

Especialización

iacedemy.com — 2026

MÓDULO 12

Arquitectura de sistemas

Nivel: Especialización

Autor: Ricardo Gutierrez

Publicación: Mayo 2026

Plataforma: iacedemy.com

API vs self-hosted

Ultimo modulo. Todo lo que has aprendido se aplica ahora a la realidad de poner IA en produccion. No en tu laptop. En un sistema real que atiende usuarios, maneja datos sensibles y cuesta dinero.

Criterios de decision

- **Datos sensibles o compliance estricto (ENS Alto, HIPAA):** self-hosted, siempre.
- **Prototipo rapido o datos publicos:** API, mas rapido.
- **Volumen mayor a 50.000 peticiones/dia:** self-hosted sale mas barato.

Calculadora de breakeven

- Haiku a 1.000 peticiones/dia: 14 EUR/mes. GPU en Hetzner: 120 EUR/mes. API gana.
- Sonnet a 1.000 peticiones/dia: 405 EUR/mes. Misma GPU: 120 EUR/mes. Self-hosted se amortiza en 9 dias.

Modelo hibrido

La respuesta real casi nunca es "todo API" o "todo self-hosted". Es hibrido:

- **Datos sensibles (SOC, GRC, PII):** vLLM self-hosted con Qwen
- **Datos no sensibles (contenido, research):** API con Sonnet
- **Clasificacion rapida:** Haiku o modelo pequeno local
- **Batch masivo:** self-hosted por coste fijo

```
Usuario pregunta sobre su cuenta
 → Haiku clasifica: es sobre datos sensibles?
   SI → Qwen self-hosted procesa con datos del cliente
   NO → Sonnet via API responde con info generica
 → En ambos casos: Qwen genera audit log
 → Datos sensibles NUNCA salen de tu red
```

Los 5 pilares de seguridad

1. Circuit breakers

```
 import time
from functools import wraps

def circuit_breaker(max_retries=3, backoff_base=2):
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            for attempt in range(max_retries):
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    if attempt == max_retries - 1:
                        raise
                    time.sleep(backoff_base ** attempt)
            raise RuntimeError("Circuit breaker activated")
        return wrapper
    return decorator
```

Max 3 reintentos con exponential backoff. Sin circuit breaker, un endpoint caido puede tumbar tu servicio completo.

2. Rate limiting

Max 100 peticiones por minuto por usuario. Protege contra uso excesivo accidental y malicioso.

3. Audit trail

Registra toda interaccion: timestamp UTC, user_id, contexto, hash del prompt, longitud de respuesta, duracion, modelo, tokens. Para ENS Alto: retener 5 anos.

4. Input sanitization

Primera linea contra prompt injection. Detectar: "ignore previous instructions", "system prompt", "forget everything". Limitar longitud. Combinar con separacion de system/user prompt, validacion de output, y principio de minimo privilegio.

5. Fallback graceful

```
 Nivel 1: IA clasifica un ticket → Falla
Nivel 2: Reglas basicas (keywords, regex) → Falla
Nivel 3: Default UNCLASSIFIED, confidence low → Humano revisara

El sistema NUNCA se detiene. Solo se degrada.
```

Costes y observabilidad

Precios por modelo (2026)

Modelo	EUR/M tokens input
Haiku	0.80
Sonnet	3.00
Opus	15.00
GPT-4o	5.00
DeepSeek V3	0.27
Qwen self-hosted	Coste GPU fijo

Ejemplo de costes mensuales

- 200 code reviews con Sonnet: 6.30 EUR
- 1.000 ticket triages con Haiku: 0.80 EUR
- 20 blog posts con Sonnet: 1.32 EUR
- 50 security audits con Opus: 15 EUR
- **Total: 29.42 EUR/mes**

5 estrategias de optimizacion

1. **Modelo correcto por tarea:** Haiku para clasificacion, Sonnet para generacion, Opus solo cuando es necesario.

2. **Prompt caching:** 90% menos por tokens de system prompt repetidos.
3. **Reducir tokens de input:** no pasar archivos enteros, resumir contextos largos.
4. **Batch processing:** agrupar peticiones similares.
5. **Self-hosted para volumen:** si haces mas de 1.000 peticiones/dia del mismo tipo.

Dashboard de observabilidad: 3 capas

- **Capa 1 (Infraestructura):** latencia p95, tasa de errores, tokens consumidos, disponibilidad.
- **Capa 2 (Calidad):** format validity, falsos positivos, satisfaccion del usuario.
- **Capa 3 (Negocio):** tickets correctamente clasificados, PRs que causaron bugs, tiempo de resolucion con vs sin IA.

Alertas recomendadas

- Latencia p95 mayor a 5 segundos
- Error rate mayor a 5%
- Tokens mayor a 2x promedio diario
- Coste mayor a 150% presupuesto

ADRs y testing

Template de ADR

Un ADR documenta una decision de arquitectura: que se decidio, por que, que alternativas se evaluaron, que trade-offs se aceptaron, y como sabras si fue correcta.

Ejemplo: ADR-042

```
 ADR-042: Modelo LLM para pipeline de code review

Contexto: 200 PRs por semana, accuracy > 85%, latencia < 10s, coste < 100
EUR/mes

Alternativa A: Opus
  92% accuracy, pero 5x mas caro. Excede presupuesto.
```

```
Alternativa B: DeepSeek V3 self-hosted
  Sin coste por token, pero 81% accuracy. Bajo threshold.

Alternativa C: Sonnet
  88% accuracy, 45 EUR/mes, sin infra propia.

Decision: Sonnet con fallback a DeepSeek para picos de volumen.
```

Eval dataset y CI gate

50 items. Accuracy global: 88%. Desglose: easy 95%, medium 88%, hard 72%. Si accuracy cae bajo threshold, el PR se bloquea en CI. Esto previene que un cambio de prompt degrade la calidad sin que nadie se de cuenta.

De prototipo a produccion en 8 semanas

Semana 1-2: Prototipo. Prompt funcional, 10-20 ejemplos probados, coste estimado, ADR con decision API vs self-hosted.

Semana 3-4: Hardening. Circuit breakers, rate limiting, input sanitization, fallback, eval dataset de 50+ items, prompts versionados.

Semana 5-6: Integracion. CI con eval gate, observabilidad con trazas y metricas, audit trail, tests de carga, runbook de incidentes.

Semana 7-8: Produccion. Deploy gradual (10%, 50%, 100%), monitoring activo 48 horas, dashboard funcional, documentacion completa.

Checklist de produccion

Gate final antes de produccion

- Modelo seleccionado con ADR documentado
- Circuit breakers en todas las llamadas a LLM
- Rate limiting por usuario
- Input sanitization activa
- Fallback graceful configurado

- Audit trail registrando toda interaccion
- Eval dataset con 50+ items y accuracy sobre threshold
- Prompts versionados con metadata
- CI ejecuta evals en cada cambio de prompt
- Observabilidad con alertas configuradas
- Coste estimado dentro de presupuesto
- Deploy gradual planificado

Ejercicio practico

Ejercicio M12: Lleva tu agente a produccion

1. Elige un agente o pipeline de los modulos anteriores.
2. Escribe un ADR con la decision de modelo (API vs self-hosted, que modelo, por que).
3. Implementa los 5 pilares de seguridad: circuit breaker, rate limit, audit trail, sanitization, fallback.
4. Crea un eval dataset de 50 items para medir accuracy.
5. Configura observabilidad: logging de 10 campos + dashboard basico.
6. Calcula el coste mensual estimado con tu volumen real.
7. Escribe el roadmap de 8 semanas para tu caso.

Bonus: Configura un deploy gradual: 10% del trafico al agente IA, 90% al sistema actual. Mide durante 48 horas.

Conclusiones clave

Key takeaways del M12

1. API vs self-hosted: depende de compliance, volumen y coste. Casi siempre es hibrido.
2. 5 pilares de seguridad: circuit breakers, rate limiting, audit trail, input sanitization, fallback graceful.
3. Costes: modelo correcto por tarea + prompt caching + batch = optimizacion sin perder calidad.
4. Observabilidad en 3 capas: infra, calidad, negocio. Con alertas configuradas.
5. ADRs documentan decisiones de IA con alternativas, trade-offs y criterios de exito.
6. 8 semanas de prototipo a produccion: prototipo, hardening, integracion, deploy gradual.

Has completado los 12 modulos core de IAcademy

Nivel 1: elegir IAs, prompts, entorno. Nivel 2: automatizar, conectar, medir. Nivel 3: especializar en dev, seguridad o negocio. Nivel 4: equipo y produccion. Ahora aplica lo aprendido en tu proyecto real.

Ir al Modulo 13

IACADEMY

iacedemy.com

De fundamentos a arquitectura de IA.
12 módulos prácticos. 24 recursos descargables.
Quizzes con certificado. Vídeos profesionales.

Empieza gratis en iacedemy.com/free