Reactor
COURSES

BROCHURE DEL PROGRAMA



ARQUITECTURA E INGENIERÍA · DISEÑO Y ESTRATEGIA DE ARQUITECTURA

Microservice Architect

Los patrones de microservicios en un mismo sistema

Decide la forma de un sistema de muchos servicios —dónde cortar, quién es dueño de cada dato, cómo se comunican, cómo fallan, cómo se despliegan y prueban por separado — combinando los patrones en un solo sistema y defendiendo con evidencia cuáles no usar.

Lo que vas a lograr Aprendes a decidir cómo se parte un sistema grande en servicios —o por qué no partirlo—, quién es dueño de cada dato y cómo hacer que los servicios se comuniquen y fallen sin arrastrar a los demás.

- Monolito modular y DDD
- Sagas y bandeja de salida
- REST, gRPC y mensajería
- Resiliencia y observabilidad
- Kubernetes y service mesh
- Pruebas de contrato

---

horas en vivo	sesiones	semanas	patrones aplicados y medidos en un mismo sistema
108	36	18	30

INICIO
16 de febrero de 2027

PRECIO
US$150 pago único · USD

MODALIDAD
Clases en vivo por Zoom · Avanzado

reactorcourses.com/programas/microservice-architect-patterns

EL PROGRAMA

Los patrones de microservicios, todos juntos y en el mismo sistema.

Decide la forma de un sistema de muchos servicios —dónde cortar, quién es dueño de cada dato, cómo se comunican, cómo fallan, cómo se despliegan y prueban por separado— combinando los patrones en un solo sistema y defendiendo con evidencia cuáles no usar. Un programa de **108 horas** para desarrolladores backend y líderes técnicos que ya operan servicios en producción. Son unos 30 patrones vistos en profundidad, cada uno aplicado y medido sobre el mismo sistema de cinco servicios que construyes de principio a fin.

Duración	108 horas · 36 sesiones	Calendario	18 semanas · 2 sesiones por semana de 3 h
Modalidad	Clases en vivo por Zoom	Grabaciones	Disponibles por 12 meses
Credenciales	3 certificados con insignia de Reactor Courses	Nivel	Avanzado
Inicio	16 de febrero de 2027	Horario	Martes y jueves, 19:00–22:00 (GMT-5)
Precio	US$150 · pago único		

Cada patrón por separado está bien documentado. Lo difícil es combinarlos: una saga cambia cómo pruebas, un service mesh cambia cómo depuras y una base por servicio cambia cómo consultas. Aquí los ves en profundidad y en relación con los demás, sobre un solo sistema que construyes de principio a fin.

01

Decidir la forma

Descomposición, datos y comunicación

Cuándo conviene un monolito modular, dónde cortar con Domain-Driven Design, quién es dueño de cada dato y cómo se coordinan los servicios sin transacciones distribuidas. Cada decisión queda en un ADR con sus consecuencias negativas aceptadas.

- ✓ Monolito modular frente a microservicios, con criterios de reversión
- ✓ Contextos delimitados, granularidad y cómo deshacer un mal corte
- ✓ Base de datos por servicio, sagas y bandeja de salida
- ✓ REST, gRPC, mensajería, gateway y contratos que evolucionan

02

Operar la forma

Resiliencia, observabilidad, despliegue y organización

Un sistema de muchos servicios falla por partes. Aprendes a contener esas fallas, a seguir una solicitud a través de cinco servicios, a desplegar uno sin tocar los demás y a repartir los servicios entre equipos sin crear dependencias imposibles.

- ✓ Tiempos de espera, reintentos con presupuesto, cortacircuitos y degradación
- ✓ Trazas distribuidas, métricas RED y depuración entre servicios
- ✓ Pipeline por servicio, Kubernetes, service mesh y canario
- ✓ Identidad entre servicios, pruebas de contrato y topologías de equipo

> Un microservicio que no se puede desplegar solo es un módulo con latencia de red.

PARA QUIÉN ES

¿Es el programa indicado para ti?

Está pensado para desarrolladores backend y líderes técnicos que ya operan servicios en producción. En 18 semanas aplicas un patrón nuevo casi en cada clase sobre un mismo sistema que crece.

Es para ti si

- ✓ Desarrolladores backend con unos tres años de oficio que ya construyeron y operaron al menos un servicio en producción.
- ✓ Líderes técnicos que tienen que decidir cómo partir un sistema y defender esa decisión ante su equipo.
- ✓ Arquitectos que quieren el mapa completo de patrones antes de profundizar en uno.
- ✓ Equipos que heredaron un «monolito distribuido» y necesitan criterios para arreglarlo.
- ✓ Ingenieros de plataforma que van a recibir decenas de servicios y quieren entender cómo se diseñan.

Te conviene otro programa

- – Quien todavía no trabajó en un sistema en producción: los patrones se entienden por los problemas que resuelven, y hay que haberlos sufrido.
- – Quien busca profundidad máxima en un solo patrón. Cerca de la mitad de los temas tiene en la escuela un programa propio que lo trata con más detalle (eventos, APIs, sistemas distribuidos, resiliencia, Kubernetes, modernización). Si ya cursaste esos programas, vas a repetir parte del contenido.
- – Quien no pueda dedicar entre 10 y 12 horas por semana durante 18 semanas, contando clases y laboratorio.
- – Quien espera que el programa recomiende microservicios siempre: varias sesiones terminan en «aquí no».
- – Quien no pueda correr contenedores en su máquina: el laboratorio necesita 16 GB de memoria.
- – Quien busque preparar un examen de certificación de un tercero: este programa no prepara para ninguno.

EL RESULTADO

Terminas con un sistema de cinco servicios y la defensa de cada patrón que usaste y que descartaste.

El proyecto final no se juzga por la cantidad de patrones que tiene. Se juzga por si cada uno resuelve un problema medido del caso, y por si sabes explicar por qué dejaste afuera los demás.

01 **ADR de la forma del sistema** Qué capacidades pasan a servicios y cuáles quedan en el monolito modular, con consecuencias negativas aceptadas y criterio de reversión (sesión 03).

02 **Monolito modular verificado** Cinco módulos con una prueba de arquitectura que hace fallar el build si alguien cruza un límite (sesión 02).

03 **Mapa de contextos del caso** Relaciones entre contextos y tres términos del glosario con modelos distintos según el contexto (sesión 05).

04 **Saga de emisión de póliza** Orquestada con Temporal y consistente aunque el cobro falle en el último paso (sesión 10).

05 **Bandeja de salida probada** El evento «póliza emitida» sale una sola vez para el consumidor aunque el proceso muera a mitad del camino (sesión 11).

06 **Comparación REST y gRPC medida** Latencia p95 y bytes por llamada bajo la misma carga, con la elección justificada para el tráfico interno (sesión 13).

07 **Flujo asíncrono con mensajes fallidos** Cola de mensajes muertos y reproceso sin duplicar efectos después de un mensaje venenoso (sesión 18).

08 **Cadena resistente a una dependencia lenta** Presupuesto de tiempo, reintentos acotados, cortacircuitos y degradación, con el p95 dentro del objetivo (sesiones 19 a 21).

09 **Traza de punta a punta** Una emisión de póliza seguida en una sola traza a través de HTTP, gRPC y Kafka (sesión 22).

10 **Canario con reversión automática** Una versión defectuosa revertida antes de superar el umbral de error acordado (sesión 27).

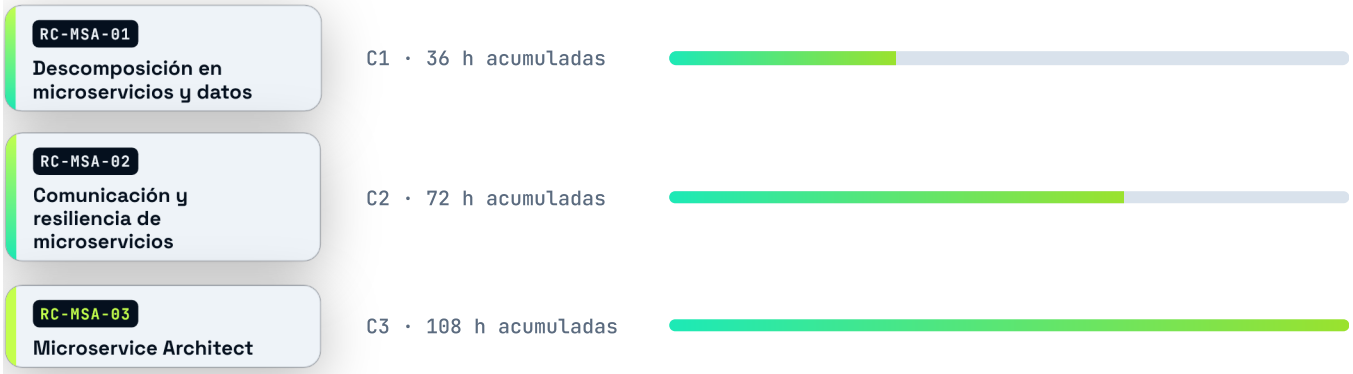
11 **Contratos que bloquean cambios** Pruebas de Pact que impiden que un proveedor rompa a su consumidor (sesión 29).

12 **Sistema final defendido** Documento de arquitectura, un ADR por patrón usado y descartado, y una falla elegida por el comité ejecutada en vivo (sesiones 34 a 36).

RUTA DE CERTIFICACIÓN

3 credenciales progresivas, no una al final.

Cada etapa cierra un bloque completo: la forma del sistema y sus datos, la comunicación y la resiliencia, y la operación y organización. Si algo te obliga a pausar a mitad del programa, ya tienes credenciales defendibles con su evidencia.



CERTIFICACIÓN INTEGRAL

Microservice Architect — Los patrones de microservicios en un mismo sistema (108 horas)

La credencial final requiere entregar el sistema integrado, su documento de arquitectura y aprobar la defensa ante el comité. Los certificados son emitidos por Reactor Courses y acreditan las horas y competencias del programa.

Cada certificado de Reactor Courses viene con su insignia, aparte de cualquier certificación oficial.

Cualquiera puede comprobar un certificado de Reactor Courses con su código en reactorcourses.com/verificar.

TEMARIO

Las 36 sesiones, abiertas y sin letra chica.

108 horas repartidas en 12 módulos. Así sabes qué verás, qué practicarás y qué entregarás en cada una de las 18 semanas.

MÓDULO 1 **CUÁNDO SÍ Y CUÁNDO NO** — 9 h · 3 sesiones

01 **Qué resuelven los microservicios y qué problemas crean**

despliegue independiente como objetivo central · autonomía de equipos y su costo de coordinación · falacias de la computación distribuida aplicadas a un caso real · latencia y disponibilidad compuestas de una cadena de llamadas · costo operativo por servicio: pipeline, monitoreo, guardia · el caso del programa: aseguradora de autos con cotización, emisión, cobranza, siniestros y notificaciones

LABORATORIO: Levantar el monolito de la aseguradora con Docker Compose y medir la línea base de latencia p95 y de tiempo de despliegue que el diseño final tendrá que mejorar o justificar

02 **El monolito modular como alternativa de primera clase**

módulos con interfaz pública y dependencias declaradas · reglas de dependencia verificadas en el build · base de datos compartida con esquemas por módulo · qué se gana y qué no frente a microservicios · señales de que un módulo ya pide ser servicio · costo de deshacer una separación prematura

LABORATORIO: Reorganizar el monolito del caso en cinco módulos y hacer fallar el build con una prueba de arquitectura cuando un módulo importa el interior de otro

03 **Decidir la forma del sistema con criterios explícitos**

drivers del caso: frecuencia de cambio, escala desigual, equipos · matriz monolito modular frente a microservicios por capacidad · registro de decisión de arquitectura (ADR) con consecuencias · presupuesto de complejidad: cuántos servicios puede operar el equipo · criterios de reversión de la decisión · costo operativo estimado por servicio adicional

LABORATORIO: Escribir el ADR que decide qué capacidades del caso pasan a servicios y cuáles quedan en el monolito modular, con al menos dos consecuencias negativas aceptadas y su criterio de reversión

MÓDULO 2 DESCOMPOSICIÓN DEL DOMINIO — 9 h · 3 sesiones

04 **Capacidades de negocio y subdominios**

mapa de capacidades de la aseguradora · subdominio central, de soporte y genérico · dónde invertir diseño y dónde comprar · flujos de valor de punta a punta: de la cotización al pago del siniestro · frecuencia de cambio por capacidad como criterio de corte · capacidades que se compran o se tercerizan

LABORATORIO: Construir el mapa de capacidades del caso clasificando cada una en central, de soporte o genérica, con la evidencia de frecuencia de cambio que justifica cada clasificación

05 **Contextos delimitados y mapa de contextos**

lenguaje ubicuo y la palabra «póliza» en cuatro contextos · relaciones: cliente-proveedor, conformista, núcleo compartido · capa anticorrupción como relación de contexto · propiedad del modelo por equipo · mapa de contextos como artefacto vivo · eventos y comandos que cruzan cada frontera de contexto

LABORATORIO: Dibujar el mapa de contextos del caso con el tipo de relación entre cada par y demostrar con tres términos del glosario que el mismo concepto tiene modelos distintos en contextos distintos

06 **Granularidad: cortar bien y deshacer un mal corte**

acoplamiento temporal, de datos y de despliegue · cohesión funcional y cambios que cruzan servicios · servicio anémico y monolito distribuido como antipatrones · métrica de cambios que tocan más de un servicio · fusionar servicios sin perder datos · regla de cambio conjunto como señal de fusión

LABORATORIO: Analizar el historial de cambios simulado de una versión mal cortada del caso, identificar los dos servicios que siempre cambian juntos y proponer la fusión con la reducción medida de cambios cruzados

MÓDULO 3 DATOS Y PROPIEDAD — 9 h · 3 sesiones

07 **Una base de datos por servicio y quién es dueño de cada dato**

propiedad del dato y una sola fuente de verdad · esquema por servicio, base por servicio y servidor por servicio · datos de referencia compartidos sin acoplar · matriz de propiedad del dato · migraciones de esquema independientes por servicio · copias de solo lectura de datos ajenos y su dueño

LABORATORIO: Separar los datos de cotización y de pólizas en bases propias de PostgreSQL y demostrar que cada servicio migra su esquema sin detener al otro

08 **Consultas que cruzan servicios**

composición de API y su latencia · réplica local de datos ajenos alimentada por eventos · vista materializada de lectura para pantallas que cruzan dominios · obsolescencia aceptable y cómo declararla · cuándo una consulta cruzada indica un mal corte · paginación y filtros en consultas compuestas

LABORATORIO: Implementar la pantalla «estado de mi póliza» de dos formas —composición de API y vista materializada— y comparar latencia p95 y obsolescencia medida bajo la misma carga

09 **Consistencia entre servicios sin transacciones distribuidas**

por qué no usar commit en dos fases entre servicios · consistencia eventual con plazos declarados · invariantes que deben cumplirse siempre y cuáles pueden esperar · reservas y estados intermedios visibles · reconciliación como red de seguridad · tiempos de convergencia medidos frente a los declarados

LABORATORIO: Clasificar las ocho invariantes del caso en inmediatas o eventuales con su plazo máximo y diseñar el estado intermedio de la emisión de póliza que las respeta

MÓDULO 4 TRANSACCIONES DE NEGOCIO ENTRE SERVICIOS

9 h · 3 sesiones

10 Sagas orquestadas y coreografiadas

saga como secuencia de transacciones locales · acciones compensatorias y lo que no se puede compensar · orquestación con un coordinador explícito · coreografía por eventos y su visibilidad · criterio de elección por número de pasos y de equipos · tiempo máximo de una saga y qué hacer al vencer

LABORATORIO: Implementar la emisión de póliza con cobro inicial como saga orquestada con Temporal y demostrar que un rechazo de la tarjeta en el último paso deja póliza y cobranza en un estado consistente

11 Bandeja de salida, idempotencia y deduplicación

escritura dual y por qué falla · bandeja de salida transaccional · consumidor idempotente con tabla de mensajes procesados · claves de idempotencia en la API pública · reintentos seguros de punta a punta · retención y limpieza de la tabla de bandeja de salida

LABORATORIO: Publicar el evento «póliza emitida» con bandeja de salida y demostrar, matando el proceso entre la escritura y la publicación, que el evento sale exactamente una vez desde el punto de vista del consumidor

12 Event sourcing y CQRS dentro de un servicio

estado como secuencia de eventos · proyecciones de lectura y su reconstrucción · costo de versionar eventos ya guardados · auditoría regulatoria como motivo legítimo · cuándo CQRS sin event sourcing alcanza · instantáneas para acelerar la reconstrucción

LABORATORIO: Evaluar los cinco servicios del caso para event sourcing con un registro de decisión e implementarlo solo en siniestros, reconstruyendo la proyección de estado desde cero con resultado idéntico

MÓDULO 5 COMUNICACIÓN SÍNCRONA

9 h · 3 sesiones

13 REST y gRPC entre servicios

contrato primero con OpenAPI y con Protocol Buffers · llamadas internas frente a API pública · tamaño de mensaje y latencia medidos · compatibilidad hacia atrás en cada formato · errores y códigos de estado coherentes · plazos y cancelación propagados en gRPC

LABORATORIO: Exponer la tarificación del caso por REST con FastAPI y por gRPC y medir latencia p95 y bytes por llamada bajo la misma carga, documentando cuál se elige para el tráfico interno y por qué

14 API gateway y backend para frontend

responsabilidades del gateway: enrutamiento, autenticación, límites · BFF por canal: web, app y corredores · lógica de negocio que no debe vivir en el gateway · agregación en el borde y su costo · un gateway por equipo o uno compartido · versionado de rutas en el gateway

LABORATORIO: Poner un gateway Envoy delante del caso y un BFF para la app móvil que agregue tres servicios, y demostrar que la app hace una sola llamada por pantalla

15 Descubrimiento de servicios y balanceo

descubrimiento en el cliente y en el servidor · DNS del clúster y sus cachés · balanceo por turno, por menor carga y por afinidad · comprobaciones de salud activas y pasivas · conexiones persistentes de gRPC y el desbalance que provocan · drenado de conexiones al desplegar una réplica

LABORATORIO: Reproducir el desbalance de conexiones gRPC persistentes entre tres réplicas de tarificación, corregirlo con balanceo por solicitud y demostrar la distribución pareja con métricas por réplica

MÓDULO 6 COMUNICACIÓN ASÍNCRONA

9 h · 3 sesiones

16 **Comandos, eventos y el broker que corresponde**

comando frente a evento: quién decide · cola con RabbitMQ frente a log con Kafka · orden por clave de partición · grupos de consumidores y paralelismo · acoplamiento oculto en eventos demasiado ricos · tamaño del evento y datos que no deben viajar

LABORATORIO: Implementar las notificaciones del caso con RabbitMQ y la bitácora de eventos de pólizas con Kafka, y justificar cada elección con una propiedad del dominio demostrada en la ejecución

17 **Contratos de eventos y evolución de esquemas**

esquema del evento como contrato público · AsyncAPI para documentar canales y mensajes · registro de esquemas con reglas de compatibilidad · cambios compatibles e incompatibles · convivencia de dos versiones de un evento · eventos con datos personales y su minimización

LABORATORIO: Registrar los esquemas del caso en Apicurio con compatibilidad hacia atrás y demostrar que el registro rechaza un cambio incompatible antes de que llegue a un consumidor

18 **Mensajes fallidos, orden y reproceso**

reintentos con espera creciente en el consumidor · cola de mensajes muertos y su operación · mensaje venenoso y cómo aislarlo · garantías de orden cuando hay reintentos · reproceso seguro de un rango de mensajes · alertas sobre la profundidad de la cola de mensajes muertos

LABORATORIO: Inyectar un mensaje venenoso en el flujo de siniestros, enviarlo a la cola de mensajes muertos sin detener el resto y reprocesarlo después de corregir el consumidor sin duplicar efectos

MÓDULO 7 RESILIENCIA ENTRE SERVICIOS

9 h · 3 sesiones

19 **Tiempos de espera y reintentos con presupuesto**

tiempo de espera por llamada derivado del objetivo de latencia · presupuesto de tiempo propagado en la cadena · reintentos con espera exponencial y fluctuación · presupuesto de reintentos para evitar tormentas · operaciones que no se deben reintentar · tiempos de espera en el cliente frente al servidor

LABORATORIO: Provocar con Toxiproxy una dependencia lenta en la cadena de emisión, medir la tormenta de reintentos y contenerla con presupuesto de tiempo y de reintentos hasta mantener el p95 dentro del objetivo

20 **Cortacircuitos y compartimentos**

estados del cortacircuitos y sus umbrales · compartimentos por dependencia: hilos y conexiones separados · límites de concurrencia adaptativos · configurar en la aplicación o en el proxy · pruebas de que el circuito abre y se recupera · métricas para elegir los umbrales del cortacircuitos

LABORATORIO: Configurar cortacircuitos y compartimentos en Envoy para la dependencia de inspección vehicular y demostrar que su caída no agota los recursos de la cotización

21 **Degradación, respaldos y límite de tasa**

funciones esenciales y prescindibles por criticidad · respaldos: valor en caché, valor por defecto, cola diferida · límite de tasa por cliente y por ruta · desprendimiento de carga bajo sobrecarga · comunicar la degradación al usuario · pruebas de carga que verifican la degradación

LABORATORIO: Degradar la cotización cuando cae el servicio de descuentos usando la última tabla vigente y demostrar con una prueba de carga que el sistema sigue cotizando dentro del objetivo con la dependencia apagada

MÓDULO 8 OBSERVABILIDAD DE UN SISTEMA DISTRIBUIDO

9 h · 3 sesiones

22 Trazas distribuidas con OpenTelemetry

contexto de traza propagado por HTTP, gRPC y mensajería · instrumentación automática y manual · muestreo en cabecera y en cola · atributos de negocio en los tramos · Collector como punto único de exportación · propagación de contexto a través del broker

LABORATORIO: Instrumentar los cinco servicios con OpenTelemetry y seguir en Jaeger una emisión de póliza completa que cruza HTTP, gRPC y Kafka en una sola traza

23 Métricas RED y registros correlacionados

tasa, errores y duración por servicio y por ruta · registros estructurados con identificador de traza · cardinalidad de etiquetas y su costo · tablero por servicio y tablero del flujo de negocio · alertas sobre síntomas, no sobre causas · objetivos de nivel de servicio por flujo de negocio

LABORATORIO: Construir en Prometheus y Grafana el tablero RED de los cinco servicios y saltar desde un error del tablero al registro y a la traza que lo explican

24 Salud, disponibilidad y depuración entre servicios

sondas de vida, de preparación y de arranque · dependencias en la sonda de preparación y sus riesgos · degradación visible en el estado de salud · método para depurar una falla que cruza servicios · registro de la investigación · reinicios en cascada por sondas mal configuradas

LABORATORIO: Resolver con reloj una falla sembrada que solo aparece en la emisión de pólizas de una provincia, partiendo del tablero y entregando la traza, el registro y la causa raíz

MÓDULO 9 DESPLIEGUE INDEPENDIENTE Y PLATAFORMA

9 h · 3 sesiones

25 Un pipeline por servicio

repositorio único o un repositorio por servicio · pipeline por servicio con pruebas, imagen y despliegue · versionado semántico de servicios y de contratos · configuración externa por entorno · dependencias de despliegue como señal de acoplamiento · artefactos inmutables promovidos entre entornos

LABORATORIO: Armar el pipeline de cada servicio del caso y demostrar que un cambio en cobranza se despliega sin reconstruir ni redesplegar ningún otro servicio

26 Kubernetes para microservicios: lo que hace falta

Deployment, Service y recursos por servicio · solicitudes y límites de CPU y memoria · autoescalado horizontal por métrica · patrón sidecar y patrón embajador · service mesh: con sidecar, sin sidecar o ninguna · presupuesto de disrupción de pods por servicio

LABORATORIO: Desplegar el caso en kind, activar el modo ambient de Istio solo en el espacio de cobranza y medir la latencia agregada frente al mismo servicio sin malla

27 Canario, azul-verde y banderas

despliegue y lanzamiento como decisiones separadas · canario con tráfico ponderado y métricas de corte · azul-verde y el problema del esquema compartido · banderas de funcionalidad con OpenFeature · reversión automática por objetivo de error · migraciones de esquema compatibles con dos versiones vivas

LABORATORIO: Publicar una versión defectuosa de tarificación como canario al 10 % y demostrar que la reversión automática actúa antes de que la tasa de error global supere el umbral acordado

MÓDULO 10 SEGURIDAD Y PRUEBAS

9 h · 3 sesiones

28 Identidad y autorización entre servicios

identidad del usuario frente a identidad del servicio · tokens de acceso propagados y su audiencia · intercambio de tokens para llamadas en nombre del usuario · TLS mutuo entre servicios · autorización por servicio con denegación por defecto · secretos y rotación de certificados de servicio

LABORATORIO: Proteger el caso con Keycloak y TLS mutuo entre servicios y demostrar con pruebas negativas que siniestros no puede llamar a cobranza aunque esté dentro de la red

29 Pruebas de componente y de contrato

pirámide de pruebas aplicada a microservicios · pruebas de componente con dependencias reales en contenedores · contratos dirigidos por el consumidor con Pact · verificación del proveedor en su pipeline · qué no debe probarse de punta a punta · estado del proveedor en las pruebas de contrato

LABORATORIO: Escribir los contratos de Pact entre el BFF móvil y pólizas, y demostrar que el pipeline de pólizas bloquea un cambio que rompería al consumidor

30 Pruebas de punta a punta mínimas y entornos efímeros

los tres recorridos de negocio que sí merecen prueba completa · datos de prueba sintéticos y reproducibles · entorno efímero por rama · pruebas en producción con tráfico sintético · costo y fragilidad de cada nivel de prueba · duración y costo del entorno efímero

LABORATORIO: Levantar un entorno efímero del caso por rama con Testcontainers y ejecutar los tres recorridos críticos en menos de diez minutos, con datos sintéticos reproducibles

MÓDULO 11 EVOLUCIÓN Y ORGANIZACIÓN

9 h · 3 sesiones

31 Ley de Conway y topologías de equipo

la estructura de equipos se refleja en la arquitectura · equipos alineados al flujo, de plataforma y habilitadores · carga cognitiva por equipo · maniobra inversa de Conway · propiedad de servicios sin dueño · contratos entre equipos y acuerdos de servicio internos

LABORATORIO: Asignar los servicios del caso a cuatro equipos con su carga cognitiva estimada y demostrar que ningún flujo de negocio central depende de más de dos equipos para un cambio típico

32 Migrar desde el monolito por partes

higuera estranguladora: interceptar y desviar · capa anticorrupción hacia el modelo heredado · extraer primero lo que más cambia · datos que viven en ambos mundos durante la migración · criterio de terminado de una extracción · enrutamiento por bandera durante la extracción

LABORATORIO: Extraer notificaciones del monolito modular detrás de una fachada con capa anticorrupción y demostrar con tráfico continuo que ningún mensaje se pierde durante el cambio

33 Gobierno de un sistema de muchos servicios

plantilla de servicio con lo mínimo obligatorio · catálogo de servicios con dueño, contratos y guardia · deprecación y retiro de una versión de API · revisiones de arquitectura livianas con ADR · métricas de entrega por servicio · servicios huérfanos y política de retiro

LABORATORIO: Crear la plantilla de servicio del caso con telemetría, salud, pipeline y contrato incluidos, y generar con ella un servicio nuevo que pasa todas las verificaciones del catálogo en el primer intento

MÓDULO 12 **PROYECTO FINAL Y DEFENSA** 9 h · 3 sesiones

34 **Integración del sistema final**

ensamblar descomposición, datos, comunicación y resiliencia · documento de arquitectura con ADR por patrón elegido · patrones descartados y por qué · verificación de la línea base de la sesión 01 · lista de verificación de entrega · costo operativo mensual estimado del diseño

LABORATORIO: Entregar el sistema del caso integrado con su documento de arquitectura, un ADR por patrón usado y otro por cada patrón descartado, y la comparación medida contra la línea base de la sesión 01

35 **Revisión cruzada y día de fallas**

revisión de la arquitectura de otro equipo con escenarios · fallas inyectadas sin aviso · detección, contención y comunicación · acciones correctivas priorizadas · registro de hallazgos recibidos y emitidos · comunicación del incidente durante el día de fallas

LABORATORIO: Responder a tres fallas inyectadas sin aviso por otro equipo en el sistema propio y entregar el informe de revisión del sistema ajeno con sus hallazgos priorizados

36 **Defensa ante el comité**

presentación de la forma del sistema y sus decisiones · preguntas del comité sobre patrones elegidos y descartados · falla elegida por el comité ejecutada en vivo · costo operativo estimado del diseño · plan de evolución a un año · riesgos aceptados y su fecha de revisión

LABORATORIO: Defender individualmente el sistema ante el comité, que elige una falla y un cambio de requisito para ejecutar en vivo

HERRAMIENTAS

Todo lo que vas a tocar con las manos.

Servicios y contratos

Python y FastAPI · gRPC y Protocol Buffers · OpenAPI · AsyncAPI

Datos

PostgreSQL · Bandeja de salida transaccional · Event sourcing en un servicio

Mensajería

RabbitMQ · Apache Kafka · Apicurio Registry

Coordinación

Temporal

Tráfico y resiliencia

Envoy · Toxiproxy · OpenFeature (flagd)

Plataforma

Docker Compose · Kubernetes con kind · Istio en modo ambient

Observabilidad

OpenTelemetry · Jaeger · Prometheus · Grafana

Seguridad

Keycloak · TLS mutuo

Pruebas

Pact · Testcontainers

Los logos son marcas de sus respectivos dueños. Se muestran solo para identificar las herramientas que se usan en el programa y no implican aval, patrocinio ni afiliación.

CÓMO FUNCIONA

Clases en vivo, un patrón por problema y el mismo sistema de principio a fin.

01 **Un solo caso para todo el programa** Una aseguradora de autos con cotización, emisión, cobranza, siniestros y notificaciones. Cada patrón entra porque el caso lo necesita, no porque toque en el temario.

02 **Problema primero, patrón después** Cada sesión empieza con una falla o una limitación medida en el sistema y termina con el patrón aplicado y su efecto medido.

03 **Un tercio de concepto, dos tercios de taller** En cada sesión de 3 horas, la primera hora es diseño y discusión y las otras dos son laboratorio sobre el caso.

04 **Decisiones escritas** Cada patrón que usas y cada patrón que descartas queda en un ADR. Al final tienes el documento de arquitectura del sistema, no una colección de ejercicios sueltos.

05 **Fallas provocadas** La resiliencia, la observabilidad y el despliegue se prueban rompiendo cosas a propósito: dependencias lentas, procesos que mueren, versiones defectuosas.

06 **Revisión entre pares y comité** Revisas el sistema de otro equipo y defiendes el tuyo ante un comité que elige la falla que se ejecuta en vivo.

Horarios · Primer grupo

Martes	19:00 - 22:00
Jueves	19:00 - 22:00

Horario referido a GMT-5 (Perú, Colombia, Ecuador y Panamá; en Ciudad de México es una hora menos).

Las clases que caigan en feriado se reprograman. El calendario definitivo se entrega al confirmar la matrícula.

REQUISITOS

Lo que necesitas traer a la sesión 01.

No hay nivelación previa. La sesión 01 asume que ya construiste servicios y que sabes leer y modificar código de otra persona.

Experiencia

- ✓ Unos tres años desarrollando backend
- ✓ Haber operado al menos un servicio en producción

Programación

- ✓ Programar con soltura en un lenguaje de backend
- ✓ Leer y escribir Python: el caso viene en Python con FastAPI

Fundamentos

- ✓ APIs REST y códigos de estado HTTP
- ✓ Bases de datos relacionales y transacciones
- ✓ Git y control de versiones

Herramientas

- ✓ Docker y Docker Compose a nivel de uso diario
- ✓ Terminal de Linux o macOS, o WSL en Windows

Equipo

- ✓ Máquina con 16 GB de memoria como mínimo
- ✓ Conexión estable para clases en vivo de 3 horas

Costos de herramientas que asumes tú `US$0 a 15 aprox.`

Estos costos NO están incluidos en el precio del programa. Todo el software del laboratorio es gratuito.

- El laboratorio corre en tu máquina con contenedores: PostgreSQL, RabbitMQ, Kafka, Temporal, Keycloak, Envoy y Kubernetes con kind, todo de código abierto
- Si tu equipo no alcanza (se recomiendan 16 GB de memoria), una máquina virtual alquilada solo durante las prácticas cuesta del orden de US$15 en todo el programa
- No se usa ningún servicio gestionado de pago

PRECIO

108 horas para decidir la forma de un sistema con evidencia, no por moda.

US$150

Pago único · USD · Incluye impuestos

Primer grupo · inicia el 16 de febrero de 2027

Incluye

- ✓ 36 sesiones en vivo por Zoom (108 horas lectivas)
- ✓ Grabación de todas las clases con acceso por 12 meses
- ✓ Repositorio del caso con el punto de partida de cada laboratorio
- ✓ Datos sintéticos y escenarios de falla para cada módulo
- ✓ Revisión de los ADR y entregables de cada módulo
- ✓ Revisión cruzada, día de fallas y defensa ante comité
- ✓ Los 3 certificados progresivos
- ✓ Canal privado del grupo para dudas entre sesiones
- ✓ Actualizaciones del material por 12 meses

No incluye

- — El equipo de trabajo: necesitas una máquina con 16 GB de memoria como mínimo
- — Servicios de nube de pago: todo el laboratorio corre en tu máquina con software gratuito
- — Exámenes ni certificaciones de terceros: este programa no prepara para ninguno

Métodos de pago

Tarjeta de crédito · Tarjeta de débito internacional · PayPal · Apple Pay · Google Pay

Garantía de 14 días. Si dentro de los primeros 14 días del programa decides que no es para ti, te devolvemos el 100 % de lo pagado. No pedimos justificación.

Paddle actúa como comerciante registrado (merchant of record): emite el comprobante y gestiona los impuestos de cada país.

PREGUNTAS FRECUENTES

Preguntas frecuentes sobre el programa.

¿Las clases son en vivo o grabadas?

En vivo por Zoom, 2 veces por semana, sesiones de 3 horas. Cada clase queda grabada y la puedes ver durante 12 meses.

¿Cuándo empiezan las clases?

El primer grupo empieza el 16 de febrero de 2027. Las clases son Martes y jueves en el horario GMT-5 (Perú, Colombia, Ecuador y Panamá; en Ciudad de México es una hora menos). Las que caigan en feriado se reprograman y el calendario definitivo se entrega al confirmar la matrícula.

¿Qué nivel necesito?

Avanzado. Necesitas unos tres años desarrollando backend y haber operado al menos un servicio en producción. Los requisitos completos están más arriba; si dudas, escríbenos a soporte@reactorcourses.com y te ayudamos.

¿Este programa se cruza con otros del catálogo?

Sí, y a propósito. Es un programa de recorrido: revisa en profundidad unos 30 patrones de microservicios y cómo se combinan. Cerca de la mitad de esos temas tiene en la escuela un programa propio que va más hondo en uno solo (eventos, APIs, sistemas distribuidos, resiliencia, Kubernetes o modernización). Aquí cada patrón se ve en dos o tres sesiones y en relación con los demás, sobre un caso que no se usa en ningún otro programa. Si ya cursaste esos programas, vas a repetir parte del contenido. No hay descuento ni convalidación entre programas.

¿Me conviene este programa o uno específico?

Si necesitas decidir la forma de un sistema completo y quieres el mapa de todos los patrones, este. Si ya sabes que tu problema es uno solo (por ejemplo, integrar sistemas por eventos), un programa específico te da más profundidad en menos horas.

¿Voy a terminar recomendando microservicios para todo?

No. El programa empieza con el monolito modular como alternativa de primera clase y varias decisiones del caso terminan en «aquí no conviene». Saber cuándo no usar un patrón es parte de lo que se evalúa.

¿En qué lenguaje se programa?

El caso viene en Python con FastAPI, porque se lee fácil aunque no sea tu lenguaje habitual. Los patrones no dependen del lenguaje: lo que se evalúa son las decisiones y su evidencia, no el dominio de un framework.

¿Tengo costos aparte?

No hay costos de nube: todo el laboratorio corre en tu máquina con contenedores y software gratuito. Solo necesitas un equipo con 16 GB de memoria como mínimo.

¿Qué pasa si falto a una clase?

Tienes la grabación y el laboratorio queda en el repositorio con su punto de partida. Los entregables de cada módulo tienen plazo, así que conviene ponerse al día antes de la clase siguiente.

¿Qué certificado recibo?

3 certificados progresivos emitidos por Reactor Courses, uno por cada bloque aprobado, y la credencial final Microservice Architect — Los patrones de microservicios en un mismo sistema (108 horas). No son un grado académico ni una certificación de ninguna organización externa.

¿Prepara para algún examen de certificación?

No. Ningún proveedor tiene un examen de arquitectura de microservicios que este programa prepare. Lo que te llevas es un sistema funcionando y su documento de arquitectura.

¿Necesito saber Kubernetes?

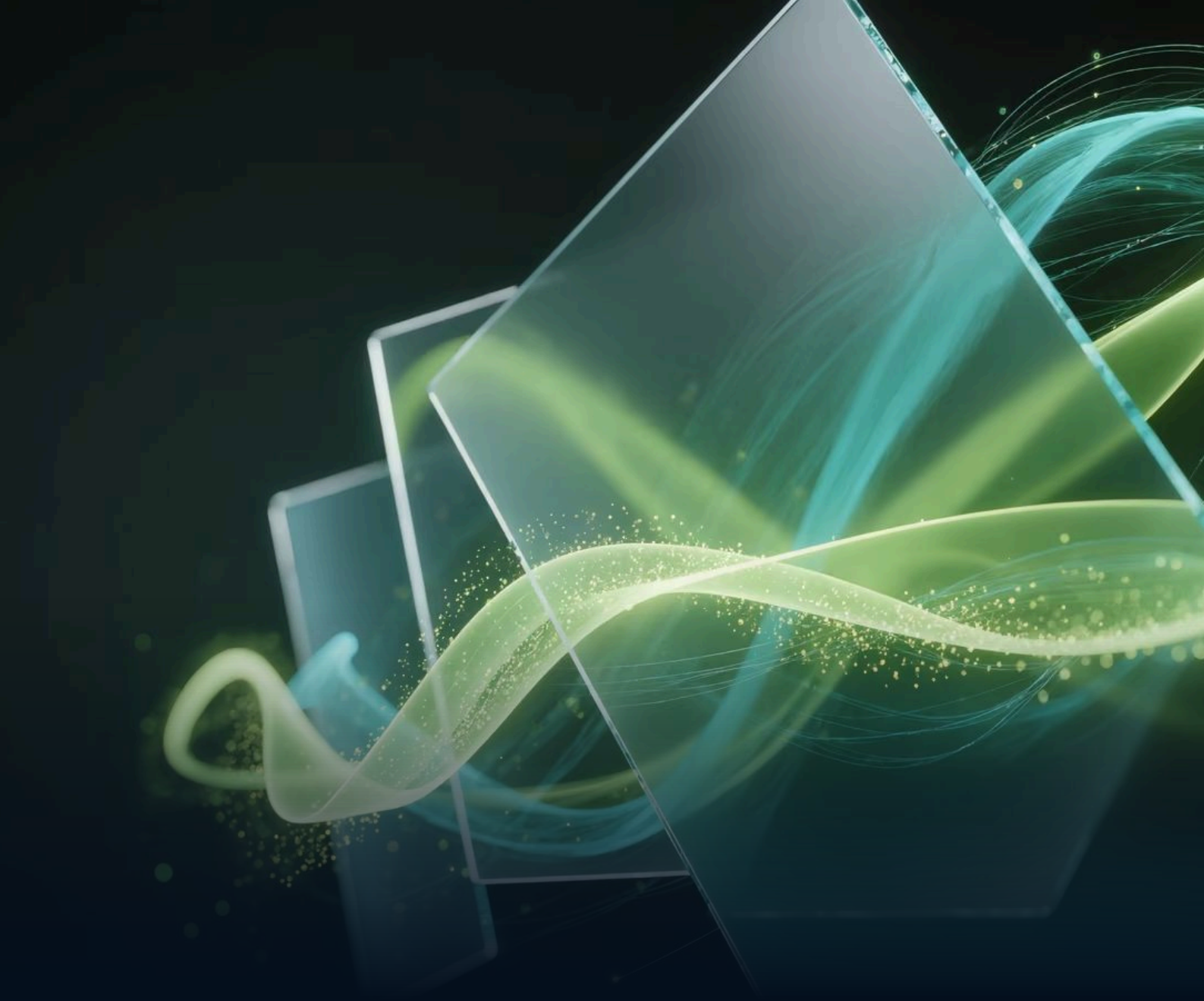
No hace falta. El módulo 9 enseña lo necesario para desplegar microservicios en Kubernetes y decidir si conviene un service mesh. Si ya lo conoces, ese módulo se apoya en lo que sabes para ir a las decisiones de diseño.

¿Cuál es la política de reembolso?

Tienes 14 días desde el inicio del programa para pedir el reembolso completo escribiendo a soporte@reactorcourses.com. Publicamos el temario completo y los requisitos para que decidas informado.

¿Cómo pago?

Con Tarjeta de crédito, Tarjeta de débito internacional, PayPal, Apple Pay, Google Pay. El precio publicado es el precio final: ya incluye los impuestos de tu país.



Reactor
COURSES

Ingeniería de software en la era de los agentes

¿Te quedó alguna duda?

Escríbenos a info@reactorcourses.com. La ficha al día, con el temario interactivo y la inscripción, está en reactorcourses.com/programas/microservice-architect-patterns.

`reactorcourses.com/programas/microservice-architect-patterns`

Información vigente al 23 de noviembre de 2026. El precio, las fechas y el temario pueden cambiar: la versión al día es la de la web.

Reactor Courses no está afiliado a Microsoft, Amazon Web Services, OpenAI, Anthropic ni Google. Los nombres de productos se mencionan únicamente con fines descriptivos del contenido académico.