Business IT

HACKATHON COPA 2026 · TRACK MICROSOFT · CHARLA 2

Operación Bajo Presión

Observabilidad y Troubleshooting en aplicaciones cloud

Cómo diagnosticar, analizar y resolver incidentes en entornos productivos utilizando Azure Monitor y Application Insights

Duración: 60 minutos

Business IT · Partner Microsoft · Azure Expert MSP

ESCENARIO

Son las 10:07. El reloj ya está corriendo.

10:00 terraform apply termina sin errores. Todo "verde".

10:03 El frontend carga, pero la lista de productos aparece vacía.

10:05 `curl -i https://.../health → HTTP/1.1 500 Internal Server Error`

10:07 Cuatro personas, cuatro teorías distintas… y ninguna evidencia.

Hoy: una metodología para pasar de "¿qué hacemos?" a "encontramos la causa raíz".

AGENDA · 60 MINUTOS

El plan de hoy



01

La aplicación falla… ¿por dónde empiezo?

Mentalidad SRE y diagnóstico por evidencia

10 min



02

Observabilidad moderna

Logs, métricas, trazas, correlación, Azure Monitor

15 min



03

Anatomía de una incidencia real

500, variables, URL, integración, CORS

15 min



04

Framework de Troubleshooting

Detectar → Observar → … → Validar

15 min

Objetivo Prepararlos para resolver los problemas que encontrarán en el hackathon con una metodología práctica de análisis basada en evidencia.

Business IT

01

La aplicación falla… ¿por dónde empiezo?

*Mentalidad de un **SRE** (Site Reliability Engineering) y diagnóstico basado en evidencia*

10 min

Business IT

Pensar como un Site Reliability Engineer



Evidencia > intuición

Toda hipótesis debe apoyarse en un dato observable: un log, un código HTTP, una traza.



El síntoma no es la causa

"No carga la lista" es lo que se ve. La causa está más abajo, y hay que bajar a buscarla.



Un cambio a la vez

Si cambias tres cosas y funciona, no sabes cuál lo arregló. Si empeora, tampoco.



Restaurar primero, perfeccionar después

El objetivo es reducir el tiempo de recuperación (MTTR), no escribir el código ideal.

Cultura blameless: se analiza el sistema, no se culpa a las personas.

Business IT

Qué hacer y qué NO hacer cuando algo falla

Hacer

- Reproducir el error y anotar la hora exacta
- Leer el mensaje de error completo, hasta el final
- Revisar Application Insights antes de tocar código
- Comparar lo que la app espera vs. lo que recibe
- Registrar cada hipótesis y su resultado
- Validar con /health y la funcionalidad real

NO hacer

- Cambiar código "a ver si funciona"
- Redesplegar sin saber qué cambiaste
- Desactivar seguridad (CORS "*") para salir del paso
- Asumir que "en mi máquina funciona" aplica en la nube
- Que los 4 del equipo investiguen lo mismo
- Declarar victoria sin validar el resultado

Business IT

Dónde está la evidencia, capa por capa

Navegador	DevTools → Console y Network: URL llamada, código HTTP, errores CORS
HTTP	curl -i / Postman: código, headers y cuerpo de la respuesta
Aplicación	Application Insights: requests, exceptions, traces, dependencies
Plataforma	App Service: Log stream, App settings, CORS, Diagnose and solve problems
Código e IaC	Repositorio: archivos de configuración, main.tf, git diff

Regla práctica

Empiecen por donde está el síntoma y bajen capa por capa hasta encontrar el primer punto donde la realidad no coincide con lo esperado.

La ausencia de evidencia también es evidencia: si el backend no registra ningún request, el problema ocurre antes de llegar a él.

Business IT

02

Observabilidad moderna

Logs, métricas, trazas, correlación de eventos, Azure Monitor y Application Insights

15 min

Business IT

Tres señales, tres preguntas



Métricas

¿QUÉ está mal?

Valores numéricos en el tiempo: tasa de error, latencia, requests/seg, CPU, memoria.

`requests · performanceCounters · customMetrics`



Trazas

¿DÓNDE está el problema?

El recorrido de una operación entre frontend, backend y dependencias, con tiempos por salto.

`requests · dependencies · operation_Id`



Logs

¿POR QUÉ ocurrió?

Eventos discretos con contexto: mensajes estructurados, excepciones y stack traces.

`traces · exceptions`

En Application Insights cada señal vive en una tabla consultable con KQL (nombres en verde).

Business IT

operation_Id: el hilo que une todo

Una sola operación, cuatro tablas

requests
GET /api/products — 500 · 212 ms

dependencies
HTTP GET catálogo interno — falló · 180 ms

exceptions
InvalidOperationException — "Setting no encontrado"

traces
Error al inicializar servicio — severityLevel 3

`operation_Id = 4bf92f35…`

KQL · correlación

```
// Todo lo que pasó en una operación
union requests, dependencies,
      exceptions, traces
| where timestamp > ago(30m)
| where operation_Id == "4bf92f35..."
| project timestamp, itemType,
          name, resultCode,
          type, message
| order by timestamp asc
```

En el portal: Failures → seleccionar request → "End-to-end transaction details" muestra esta misma cadena de forma visual.

Business IT

La plataforma y su especialista en aplicaciones

Azure Monitor

Plataforma unificada de monitoreo de Azure

Métricas de plataforma

CPU, memoria, HTTP 5xx del App Service

Activity Log

Quién cambió qué recurso y cuándo

Log Analytics

Almacén consultable con KQL

Alertas y Dashboards

Notificar y visualizar

Application Insights (APM)

Telemetría de la aplicación: requests, dependencias, excepciones, trazas, Application Map y Live Metrics.

¿Cuál uso?

Pregunta	Herramienta
¿Por qué falla este endpoint?	App Insights · Failures
¿Qué excepción y en qué línea?	App Insights · Exceptions
¿Está saturado el servidor?	Métricas App Service
¿Quién cambió la config?	Activity Log
¿Qué pasa ahora mismo?	Live Metrics / Log stream

Business IT

Lo que tendrán abierto durante el reto

Failures

Requests fallidos agrupados por código y operación. Punto de partida para un 500.

Transaction search

Buscar un request puntual y abrir su detalle end-to-end.

Live Metrics

Telemetría segundo a segundo: ideal para validar un redeploy.

Logs (KQL)

Consultas libres sobre requests, exceptions, traces y dependencies.

Application Map

Mapa de componentes y llamadas fallidas entre ellos.

Azure Insights

Vista consolidada del equipo: errores, latencia y disponibilidad.

Nota: la telemetría puede tardar 1–3 minutos en aparecer en Logs. Live Metrics y Log stream son casi inmediatos.

Business IT

Cuatro consultas que resuelven el 80% de los casos

requests

```
// 1. ¿Qué está fallando?
requests
| where timestamp > ago(30m) and success == false
| summarize total = count() by name, resultCode
| order by total desc
```

exceptions

```
// 2. ¿Qué excepción y con qué mensaje?
exceptions
| where timestamp > ago(30m)
| summarize total = count() by type, outerMessage
| order by total desc
```

traces

```
// 3. ¿Qué dijo la app? (logs estructurados)
traces
| where timestamp > ago(30m) and severityLevel >= 2
| project timestamp, severityLevel, message
| order by timestamp desc
```

dependencies

```
// 4. ¿Falla o tarda algo externo?
dependencies
| where timestamp > ago(30m)
| summarize fallos = countif(success == false),
            p95 = percentile(duration, 95) by target
```

Business IT

03

Anatomía de una incidencia real

API 500, variable de entorno, URL mal configurada, integración y CORS

15 min

Business IT

Del síntoma a la primera evidencia

Caso	Síntoma visible	Primera evidencia	Dónde mirar
API responde 500	El endpoint devuelve error genérico	Excepción con tipo, mensaje y stack	App Insights · Failures / exceptions
Variable de entorno	La app arranca pero un servicio no se inicializa	"Required setting '...' not found"	exceptions + App settings + main.tf
URL mal configurada	Frontend carga, datos vacíos	Network: host inexistente / DNS error	DevTools · Network; backend sin requests
Error de integración	404, 415 o datos mal mapeados	Ruta o campo JSON distinto al contrato	Network + requests (resultCode)
CORS	Postman funciona, navegador no	"blocked by CORS policy" en Console	DevTools · Console + requests OPTIONS

Guarden esta tabla: es su mapa de navegación durante el reto.

Business IT

Un 500 avisa que algo falló — no dice qué

Paso 1 · reproducir

```
$ curl -i https://team-47-api.azurewebsites.net/api/products
HTTP/1.1 500 Internal Server Error
Content-Type: application/problem+json
x-ms-request-id: 4bf92f35-...
```

Paso 2 · leer la excepción

```
// Application Insights → Failures → Exceptions
System.InvalidOperationException:
  Catalog service is not initialized.
   at GetProductsQueryHandler.Handle(...)
      in Application/Products/Queries/...:line 21
   at MediatR.Mediator.Send(...)
   at ProductsController.Get(...)
```

Cómo leer un stack trace

1. **Tipo**
   Qué clase de error (InvalidOperation, NullReference, Timeout…)
2. **Mensaje**
   La pista humana: casi siempre nombra lo que falta
3. **Primera línea propia**
   El primer archivo de su código: ahí empieza la búsqueda
4. **Contexto**
   ¿Qué necesita ese código para funcionar? Config, dependencia, dato

El 500 es el síntoma. La excepción es la pista.

Business IT

El frontend llama a un backend que no existe



Evidencia

Network tab: la URL solicitada no es la del App Service del equipo. En App Insights del backend: silencio total.

Causa típica

Archivo de ambiente de producción con un host genérico o copiado de desarrollo.

Trampa

En apps web, la URL se incrusta al compilar: cambiar un App Setting no basta. Hay que editar, reconstruir y redesplegar el frontend.

Business IT

Ambos lados funcionan… pero no se entienden

Desajuste	Ejemplo	Evidencia	Código típico
Ruta	`/api/product vs /api/products`	Network + requests	404
Verbo	POST donde se espera GET	requests (name)	405
Content-Type	Envía texto, espera JSON	Response body	415 / 400
Campo JSON	`productName vs name`	Response vs contrato	200 con datos vacíos
Protocolo	Frontend HTTPS llama a HTTP	Console: mixed content	Bloqueado

Los códigos 4xx apuntan al cliente (frontend); los 5xx al servidor (backend).



El contrato es el juez

Comparen la petición real (Network → Headers / Payload) contra el contrato OpenAPI. El primer campo que no coincide es su causa raíz.

Un 200 con datos vacíos también es un error.

Business IT

Postman funciona, el navegador no

Preflight

El navegador envía OPTIONS preguntando si el origen está permitido.

Respuesta

El backend responde sin Access-Control-Allow-Origin para ese origen.

Bloqueo

El navegador descarta la respuesta y muestra el error en Console.

DevTools · Console

```
Access to fetch at 'https://team-47-api...' from origin
'https://team-47-frontend...' has been blocked by CORS policy
```

Origen = protocolo + host + puerto. Debe coincidir exacto: sin "/" final, con https.

Checklist CORS

- Se corrige en el backend, no en el frontend
- Permitir el origen exacto del frontend del equipo
- Nunca "*" con credenciales: no es una solución
- No mezclar CORS del App Service (plataforma) con CORS del código: el de plataforma toma el control
- Validar en el navegador, no en Postman
- En App Insights: buscar requests OPTIONS

Business IT

"Responde" no significa "está listo"

`/health`

Liveness

¿El proceso está vivo?

`/health/deep`

Readiness

¿Configuración y dependencias listas para producción?

`/api/status`

Validación final

¿El negocio declara la release como lista (feature flags)?

Arreglar un error puede revelar el siguiente: los errores de arriba enmascaran a los de abajo.



Qué esperar en el reto

- Algunos problemas son independientes y se pueden atacar en paralelo
- Otros solo aparecen cuando los anteriores están resueltos
- Después de cada fix: redeploy y volver a observar
- Leer los logs también en modo "todo parece bien": los warnings dan pistas

Business IT

04

Framework de Troubleshooting

Una metodología de seis pasos para encontrar la causa raíz rápido

15 min

Business IT

Seis pasos, siempre en el mismo orden

Detectar

¿Qué falla, desde cuándo y a quién afecta?

Observar

Recolectar evidencia sin cambiar nada todavía.

Analizar

Correlacionar señales y ubicar la capa que falla.

Hipótesis

Una causa concreta y verificable, con su evidencia.

Corregir

Un solo cambio, en el código versionado.

Validar

Redeploy y confirmar con datos que se resolvió.

Si la validación falla, se vuelve a Observar — no a Corregir.

Una hipótesis refutada también es progreso: descarta una causa y acota la búsqueda. Lo que nunca es progreso es un cambio sin hipótesis.

Business IT

El framework en acción: un caso de CORS en 6 pasos

1

Detectar

La lista de productos está vacía en el navegador.

2

Observar

Console: "blocked by CORS policy". curl -i a la API devuelve 200.

3

Analizar

El backend responde; el navegador bloquea. La capa es seguridad web.

4

Hipótesis

"El origen permitido no coincide con la URL del frontend, porque el error nombra ese origen."

5

Corregir

Actualizar los orígenes permitidos en la configuración del backend → commit → redeploy.

6

Validar

La UI muestra productos; en Console no hay errores; /health = 200.

Business IT

Lo que no se escribe, se repite

Hora	Síntoma	Evidencia	Hipótesis	Cambio	Resultado
10:12	/api/products → 500	Exception: setting no encontrado	Nombre de variable distinto en IaC	Renombrar clave en main.tf	**Resuelto**
10:31	Lista vacía en UI	Network: host inexistente	URL de producción incorrecta	Editar env + rebuild	**Resuelto**
10:44	/health/deep degradado	traces: chequeo interno no pasa	Configuración incompleta para producción	—	**En análisis**

Evita repetir

No se prueba dos veces la misma hipótesis descartada.

Coordina al equipo

Todos ven qué se investigó y qué falta.

Alimenta CTFd

Síntomas, causas y archivos ya están documentados para responder.

Business IT

“

Los mejores equipos no son los que cometen menos errores. Son los que encuentran la causa raíz más rápido.

Mensaje final · Charla 2

Detectar · Observar · Analizar · Hipótesis · Corregir · Validar
¿Preguntas? ¡Nos vemos en el Hackathon!