Intel DevOps Assessment

Documento de Arquitectura - AI Agent

1. Resumen Ejecutivo

Este documento describe la arquitectura propuesta para un agente de IA que asiste con tareas operacionales en el entorno de monitoreo de la Online Boutique desplegada en GKE. El agente se integra con Prometheus y Grafana para analizar métricas en tiempo real, detectar anomalías, responder preguntas sobre el estado del sistema, y proporcionar recomendaciones de optimización automáticas. El diseño se basa en una arquitectura modular que permite escalar cada componente independientemente según las necesidades del equipo. Este documento es un diseño de arquitectura; no incluye la implementación completa del agente.

2. Escenarios Soportados

El agente de IA está diseñado para cubrir tres escenarios operacionales principales que cubren las necesidades diarias de un equipo de DevOps:

Escenario	Descripción	Ejemplo de interacción
Análisis de métricas	El agente monitorea las métricas recopiladas por Prometheus de forma continua y detecta anomalías o patrones inusuales automáticamente, sin necesidad de que un humano revise los datos.	CPU usage spiked 40% at 2pm, correlating with increased checkout service requests. Recommend investigating recent deployment.
Asistente de despliegue	El agente responde preguntas en tiempo real sobre el estado del sistema, la configuración actual y los pasos necesarios para resolver problemas comunes.	Usuario: Which services are consuming the most memory? / Agente: recommendationservice está usando 220Mi de los 450Mi permitidos (49%).
Recomendaciones automáticas	El agente analiza las métricas históricas y el estado actual del sistema para generar recomendaciones de optimización que el equipo pueda implementar.	Frontend service memory usage trending up 15% over 2h. Consider scaling from 1 to 2 replicas to distribute load.

3. Stack Tecnológico Propuesto

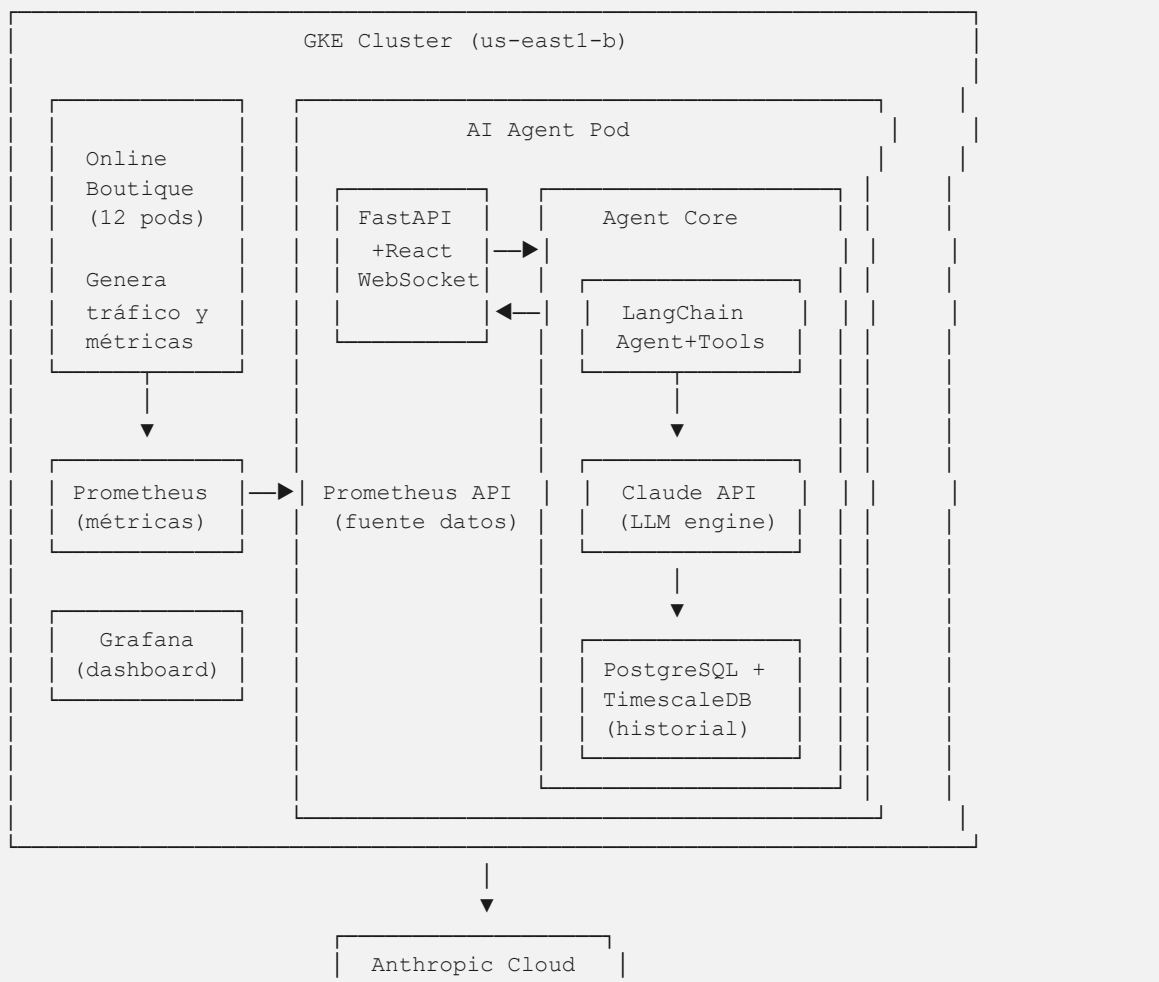
Cada tecnología fue seleccionada considerando su compatibilidad con el entorno existente (GKE, Prometheus, Grafana), el tamaño del ecosistema de desarrollo, y la facilidad de mantenimiento a largo plazo.

Capa	Tecnología	Justificación
Lenguaje principal	Python 3.11+	Ecosistema más maduro para integración con LLMs y librerias de procesamiento de datos. Compatible con las APIs de Prometheus y Kubernetes.
LLM / AI Provider	Anthropic Claude API (claude-sonnet-4-5)	Alto rendimiento en análisis de datos estructurados y razonamiento lógico. API clara y bien documentada con soporte para tool use.
Framework de agente	LangChain	Framework open-source para construir agentes con tools, memory y chains. Conecta el LLM con fuentes de datos externas de forma modular.
API de métricas	Prometheus HTTP API	API nativa de Prometheus para hacer consultas PromQL programáticamente. Permite obtener métricas históricas y en tiempo real.
API de Kubernetes	kubernetes Python client (official)	Cliente oficial para interactuar con el clúster GKE. Obtiene estado de pods, servicios y otros recursos.
Base de datos	PostgreSQL + TimescaleDB	Base relacional con extensión de series temporales. Almacena métricas históricas procesadas y contexto de conversaciones anteriores.
Backend API	FastAPI	Framework Python de alto rendimiento para APIs. Soporta WebSocket para comunicación en tiempo real y es fácil de desplegar en contenedores.
Frontend	React	Librería de interfaz de usuario que permite construir la experiencia de chat interactiva con actualizaciones en tiempo real.

Autenticación	OAuth 2.0 + JWT	Estándar ampliamente soportado. JWT permite validar tokens sin consultar una base de datos en cada request.
Contenedorización	Docker + Kubernetes	El agente se despliega como un contenedor dentro del mismo clúster GKE, garantizando consistencia con el entorno existente.

4. Diagrama de Arquitectura del Agente

El diagrama muestra los componentes del agente y cómo se integran con la infraestructura existente. El agente se despliega dentro del mismo clúster GKE para tener acceso directo a la red interna.



5. Descripción de Componentes

Agent Core (LangChain)

Es el cerebro del sistema. LangChain coordina el flujo de la conversación, decide qué tool usar según la pregunta del usuario, y combina las respuestas en un mensaje coherente. Utiliza el modelo Claude como motor de razonamiento para interpretar las métricas y generar insights accionables.

Tools (Herramientas del agente)

Son las funciones que el agente puede ejecutar. Cada tool tiene un nombre y una descripción que el LLM usa para decidir cuándo usarla. Las tools propuestas son: (1) query_prometheus — hace consultas PromQL a Prometheus para obtener métricas en tiempo real, (2) get_pod_status — obtiene el estado actual de los pods usando el Kubernetes Python client, (3) get_historical_metrics — consulta métricas históricas de TimescaleDB, (4) detect_anomalies — compara métricas actuales contra baselines históricos para detectar desviaciones.

Prometheus API Integration

El agente se conecta con la API HTTP de Prometheus para hacer consultas PromQL programáticamente. Esto permite obtener las mismas métricas que se ven en Grafana pero de forma estructurada que el LLM puede procesar y analizar.

Kubernetes Python Client

Permite al agente interactuar directamente con el clúster GKE para obtener información sobre pods, servicios, nodos y otros recursos. Esencial para responder preguntas sobre el estado del despliegue.

PostgreSQL + TimescaleDB

Almacena dos tipos de datos: métricas históricas procesadas que permite hacer análisis de tendencias y detectar anomalías comparando el estado actual contra el histórico, y el

historial de conversaciones para que el agente recuerde el contexto de interacciones anteriores.

FastAPI + WebSocket + React

FastAPI expone la API del agente. WebSocket mantiene una conexión en tiempo real entre el navegador y el backend, permitiendo que las respuestas aparezcan de forma streaming (palabra por palabra). React renderiza la interfaz web interactiva donde el usuario hace preguntas y ve las respuestas.

6. Flujo de Datos

El flujo de datos del agente sigue estos pasos cuando un usuario hace una pregunta:

Paso	Descripción
1. Input del usuario	El usuario escribe una pregunta en la interfaz web (React). El mensaje se envía al backend (FastAPI) mediante WebSocket.
2. Procesamiento inicial	FastAPI recibe el mensaje y lo pasa al Agent Core (LangChain). LangChain agrega el contexto de conversaciones anteriores almacenado en PostgreSQL.
3. Decisión del LLM	El mensaje junto con el contexto se envía al modelo Claude (Anthropic API). El modelo analiza la pregunta y decide qué tools necesita usar para responder.
4. Ejecución de tools	LangChain ejecuta las tools seleccionadas. Por ejemplo, si el usuario pregunta sobre CPU, se ejecuta query_prometheus con la consulta PromQL apropiada.
5. Síntesis de respuesta	Los resultados de las tools se envían de vuelta al modelo Claude, que sintetiza la información y genera una respuesta coherente y útil.
6. Respuesta al usuario	La respuesta se envía al frontend por WebSocket y aparece en la interfaz. El historial de conversación se guarda en PostgreSQL.

7. Puntos de Integración

El agente se integra con los componentes existentes del entorno sin requerir cambios significativos en la infraestructura actual:

Componente existente	Punto de integración	Protocolo
Prometheus	HTTP API en puerto 9090	REST con consultas PromQL
Grafana	Sin integración directa	El agente usa las mismas métricas pero las obtiene directamente de Prometheus
GKE Cluster	Kubernetes API Server	HTTPS con service account token
Online Boutique pods	Indirecto mediante Prometheus	Las métricas de los pods se obtienen a través de Prometheus

8. Interfaz de Usuario

La interfaz propuesta es una aplicación web tipo chat donde el usuario puede hacer preguntas en lenguaje natural y el agente responde con análisis, insights y recomendaciones. La interfaz tiene tres secciones:

Sección	Descripción
Panel de chat	Área principal donde el usuario escribe preguntas y ve las respuestas del agente. Las respuestas aparecen de forma streaming usando WebSocket.
Panel de métricas	Visualizaciones embebidas que muestran las métricas actuales más relevantes (CPU, memoria). Se actualizan en tiempo real.
Panel de alertas	Muestra las anomalías detectadas por el agente automáticamente. Cada alerta incluye una explicación de qué se detectó y por qué es importante.

9. Pseudocódigo

9.1 Definición de Tools del Agente

```
# Definición de las tools que el agente puede usar
from langchain.tools import tool
```

```
import requests

@tool
def query_prometheus(promql_query: str) -> dict:
    """Ejecuta una query PromQL contra Prometheus y retorna los resultados."""
    response = requests.get(
        'http://prometheus:9090/api/v1/query',
        params={'query': promql_query}
    )
    return response.json()

@tool
def get_pod_status(namespace: str = 'default') -> list:
    """Obtiene el estado actual de todos los pods en un namespace."""
    v1 = kubernetes.client.CoreV1Api()
    pods = v1.list_namespaced_pod(namespace=namespace)
    return [
        {'name': pod.metadata.name, 'status': pod.status.phase,
         'restarts': pod.status.container_statuses[0].restart_count}
        for pod in pods.items
    ]

@tool
def detect_anomalies(metric: str, threshold: float = 2.0) -> dict:
    """Detecta anomalías comparando métricas actuales contra el baseline
histórico."""
    current = query_prometheus(f'rate({metric}[5m])')
    historical_avg = get_historical_average(metric, period='24h')
    if current > historical_avg * threshold:
        return {'anomaly': True, 'current': current,
                'baseline': historical_avg, 'deviation': current / historical_avg}
    return {'anomaly': False}
```

9.2 Inicialización del Agente

```
# Configuración y ejecución del agente
from langchain.agents import create_openai_tools_agent, AgentExecutor
from langchain_anthropic import ChatAnthropic

# Inicializar el modelo LLM
llm = ChatAnthropic(
    model='claude-sonnet-4-5',
    api_key=os.environ['ANTHROPIC_API_KEY']
)

# Registrar las tools disponibles
tools = [query_prometheus, get_pod_status, detect_anomalies]

# Crear el agente
agent = create_openai_tools_agent(llm=llm, tools=tools, prompt=system_prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

# Ejecutar cuando el usuario hace una pregunta
def handle_user_question(question: str, conversation_history: list) -> str:
    response = agent_executor.invoke({
        'input': question,
```

```
        'chat_history': conversation_history
    })
    save_to_history(conversation_history, question, response['output'])
    return response['output']
```

9.3 Endpoint de la API (FastAPI + WebSocket)

```
# Backend API con streaming de respuestas
from fastapi import FastAPI, WebSocket

app = FastAPI()

@app.websocket('/ws/chat')
async def websocket_chat(websocket: WebSocket):
    await websocket.accept()
    conversation_history = load_history(websocket.session_id)

    while True:
        question = await websocket.receive_text()

        # Streaming: enviar la respuesta palabra por palabra
        async for chunk in agent_executor.astream({
            'input': question,
            'chat_history': conversation_history
        }):
            await websocket.send_text(chunk)

        # Guardar en historial
        save_to_history(conversation_history, question, full_response)
```

10. Consideraciones de Seguridad

Área	Consideración	Implementación
API Keys	La clave de Anthropic API no debe estar en el código	Se almacena como un Secret de Kubernetes y se inyecta como variable de entorno al pod del agente.
Autenticación	Solo usuarios autorizados deben poder usar el agente	OAuth 2.0 con JWT para autenticar cada request al backend.
Comunicación	Los datos en tránsito deben estar cifrados	TLS en todas las comunicaciones: entre frontend y FastAPI, y entre el agente y las APIs externas.
Kubernetes RBAC	El agente solo debe tener permisos mínimos necesarios	Se crea un ServiceAccount específico con permisos de solo lectura sobre pods y servicios.
Datos sensibles	Las métricas no deben filtrar información de producción	El agente solo tiene acceso al namespace "default" donde

		está la Online Boutique.
Rate limiting	Prevenir abuso de la API del agente o de Anthropic	Se implementa rate limiting en FastAPI y se monitorea el uso de tokens de la API de Anthropic.

11. Consideraciones de Escalabilidad

Aspecto	Propuesta
Múltiples instancias	Desplegar múltiples réplicas del pod del agente detrás de un load balancer para manejar más usuarios.
Caché de métricas	Cachear métricas recientes en memoria (Redis) para evitar demasiadas consultas a Prometheus.
Procesamiento asíncrono	Usar una cola de mensajes (RabbitMQ o Pub/Sub de GCP) para procesar preguntas de forma asíncrona.
Histórico de métricas	TimescaleDB escala automáticamente con particionamiento por tiempo, permitiendo meses de métricas.
Multi-cloud	La arquitectura es independiente del proveedor. El mismo agente puede conectarse a clústeres en AWS (EKS) o Azure (AKS).

12. Consideraciones de Costos

Componente	Costo estimado	Optimización
Anthropic Claude API	$0.01 - $0.05 por conversación	Minimizar el contexto enviado al LLM. Usar caché para preguntas frecuentes. Usar modelos más pequeños (Haiku) para queries simples.
GKE (pod del agente)	~$20-40/mes por réplica	Usar spot instances cuando sea posible. Escalar automáticamente según demanda.
PostgreSQL + TimescaleDB	~$50-100/mes	Implementar políticas de retención de datos. Comprimir métricas antiguas.
Almacenamiento	Según cantidad de métricas	Agregar métricas antiguas (summarize) para reducir el espacio total.

13. Resumen

El agente de IA propuesto en este documento complementa el stack de monitoreo existente (Prometheus + Grafana) añadiendo una capa de inteligencia artificial que puede analizar métricas, detectar anomalías automáticamente, responder preguntas en lenguaje natural sobre el estado del sistema, y generar recomendaciones de optimización. La arquitectura es modular, segura y escalable, permitiendo que cada componente se adapte independientemente según los requisitos del equipo. El diseño está enfocado en ser práctico y fácil de implementar dentro de la infraestructura GKE ya existente, minimizando los cambios necesarios en el entorno actual.