BIENVENIDOS

aws Cloud Clubs

aws Student Community Day

Medellín

Cloud Clubs IUDIGITAL

Cloud Clubs UD

Cloud Clubs UDEA

Cloud Clubs EAN

aws Women Colombia User Group

Cloud Clubs aws Student Community Day Medellín

Creación de API preparadas para IA: implementación de MCP en AWS para una integración fluida con IA

Daniel Alejandro Figueroa Arias

Engineering Manager
AWS Community Builder
Alejofig.com
2025-06-28

¿Por qué esta charla?

- La IA está cambiando la forma en que interactuamos con APIs.
- Necesitamos APIs preparadas para colaborar con agentes.

¿Qué es un LLM (Large Language Model)?

1. Un LLM es un modelo de IA entrenado con enormes cantidades de texto para comprender y generar lenguaje natural.

1. Ejemplos: GPT, Claude (Anthropic), Llama, etc.

1. Se usan para tareas como generación de texto, resumen, traducción y como “cerebro” de agentes.

¿Qué es MCP (Model Context Protocol)?

1. Protocolo abierto de Anthropic para estandarizar cómo los modelos de IA consumen y exponen contexto.

1. Facilita que agentes y modelos compartan datos estructurados (esquemas, metadatos) de forma consistente.

1. Permite describir tus endpoints y su semántica para que los modelos sepan cómo llamarlos.

Objetivo

- Mostrar cómo convertir cualquier API a “IA-ready” usando MCP.

- Demostrar flujo end-to-end con AWS y FastMCP.

Objetivo

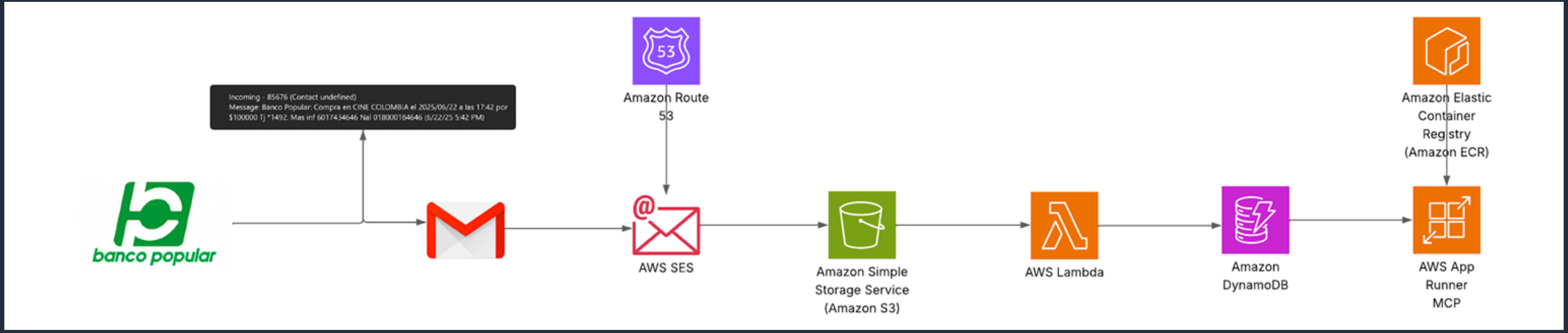
- Mostrar cómo convertir cualquier API a “IA-ready” usando MCP.

- Demostrar flujo end-to-end con AWS y FastMCP.

¿De nuevo, por qué esta charla?

1. Empoderar a los Modelos de lenguaje natural LLMs dándoles herramientas para que trabajen con su capacidad de “razonar”, permite optimizar los flujos de trabajo.

Caso de uso real



Demostración convertir API a MCP

```python
from core.config import settings
import asyncio
import logging
from fastmcp import FastMCP
from fastmcp.server.openapi import RouteMap, RouteType

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
)

app = FastAPI(…

# CORS configuration
app.add_middleware(…

# Health check endpoint (no auth required)
@app.get("/", tags=["health"])
async def health():…


# Include API routes without API key dependency
app.includ
    (function) async def check_mcp(mcp: Any) -> Any

async def check_mcp(mcp: FastMCP):…

custom_maps = […

if __name__ == "__main__":
    async def main():
        mcp = FastMCP.from_fastapi(app=app, route_maps=custom_maps)
        await check_mcp(mcp)
        await mcp.run_async(transport="sse", host="0.0.0.0", port=8000)  # Usa

    asyncio.run(main())
```

Despliegue en AWS

ghain-api > Dockerfile

```dockerfile
FROM python:3.12-slim

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

ENV PORT 8080

WORKDIR /app

RUN pip install --no-cache-dir uv

COPY pyproject.toml uv.lock* ./
COPY requirements.txt ./


RUN uv pip install -r requirements.txt --system


COPY . .


EXPOSE 8080

CMD ["python",  "app/main.py"]
```

main.tf

berghain-api > main.tf

```hcl
provider "aws" {
  region = "us-east-1" # Ej: "us-east-1"
}

# --- Variables (Considera usar variables.tf y terraform.tfvars para producci
variable "ecr_image_uri" {
  description = "El URI completo de la imagen Docker en ECR (ej: 123456789012.
  type        = string
}

# --- IAM Role para la Instancia de App Runner ---
data "aws_iam_policy_document" "apprunner_instance_assume_role_policy" {
  statement {
    actions = ["sts:AssumeRole"]
    principals {
      type        = "Service"
      identifiers = ["tasks.apprunner.amazonaws.com"]
    }
  }
}

resource "aws_iam_role" "apprunner_instance_role" {
  name               = "AppRunnerInstanceRole-BerghainAPI"
  assume_role_policy = data.aws_iam_policy_document.apprunner_instance_assume_
  inline_policy {
    name = "AppRunnerInstancePermissions"
    policy = jsonencode({
      Version = "2012-10-17"
      Statement = [
        {
          Effect   = "Allow",
          Action   = "dynamodb:*",
          Resource = "*"
        }
      ]
    })
  }
}
```

Uso con un agente

```python
import asyncio
import logging
from pydantic_ai import Agent
from pydantic_ai.mcp import MCPServerHTTP # Corrected import path
from dotenv import load_dotenv
load_dotenv()
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("MCP_TEST")


async def test_pydantic_mcp():
    """Test Pydantic AI MCP integration with sanitized server."""
    logger.info("Testing Pydantic AI MCP integration")
    mcp_server = MCPServerHTTP(url="https://mcp.alejofig.com/sse") # Change to

    agent = Agent(
        "openai:gpt-4.1-mini",
        system_prompt="You are a helpful assistant that can answer questions a
        mcp_servers=[mcp_server]
    )

    user_query = "Dame las compras que he hecho en la ultima semana"

    async with agent.run_mcp_servers():
        logger.info(f"Executing query: {user_query}")
        result = await agent.run(user_query)
        print(result)
        logger.info(f"Got result: {result.output}")

async def main():
    await test_pydantic_mcp()

if __name__ == "__main__":
    asyncio.run(main())
```

Demo

Conclusiones

1. API “IA-ready” en minutos: Con FastMCP, transformas tu FastAPI en una herramienta MCP consumible por LLMs con muy poco código.

1. Integración fluida: Los esquemas MCP estandarizados permiten a agentes (Claude, Bedrock...) descubrir y usar tus endpoints sin esfuerzo.

1. Escalabilidad segura: Despliegue serverless en AWS garantiza crecimiento elástico y protección de datos.

1. Nuevas oportunidades: Copilotos financieros, reportes automáticos y orquestación multi-agente son solo el comienzo.

aws Cloud Clubs
aws Student Community Day
Medellín

Thank you!