

Guía para Desarrollo Local de Agentes de IA con Google Agent Development Kit

12 de enero de 2026

1. Requerimientos del Sistema

1.1. Requerimientos de Hardware Mínimos y Recomendados

Componente	Mínimo	Recomendado
Procesador (CPU)	Intel i5 de 8 ^a generación o AMD equivalente	Intel i7/i9 de 11 ^{a+} generación o AMD Ryzen 7/9
Memoria RAM	16 GB DDR4	32 GB DDR4 o superior
Almacenamiento	256 GB SSD (50 GB libres)	512 GB NVMe SSD o superior
GPU (Opcional)	Integrada	NVIDIA RTX 3060+ con 8+ GB VRAM (para modelos locales)
Sistema Operativo	Windows 10, macOS 10.15+, Ubuntu 20.04+	Ubuntu 22.04 LTS o Windows 11 con WSL2
Conexión Internet	Banda ancha para descargas	Estable para APIs en la nube

Cuadro 1: Especificaciones de hardware para desarrollo local

1.2. Requerimientos de Software Esenciales

■ Sistema Operativo:

- Windows 10/11 con WSL2 (Windows Subsystem for Linux 2)
- Ubuntu 20.04 LTS o superior (recomendado)
- macOS 12 Monterey o superior

■ Entorno Python:

- Python 3.9, 3.10 o 3.11 (3.12 puede tener incompatibilidades)
- pip 23.0 o superior
- virtualenv o conda para manejo de entornos

■ Herramientas de Desarrollo:

- Git 2.30+ para control de versiones
- Docker Desktop (opcional, para contenedores)
- Visual Studio Code o PyCharm IDE

■ Google Cloud SDK:

- CLI de gcloud instalada y configurada
- Cuenta de Google Cloud Platform activa

- Proyecto GCP creado con facturación habilitada
- Controladores GPU (Opcional):
 - NVIDIA CUDA Toolkit 11.8 o 12.0
 - cuDNN compatible con la versión de CUDA

2. Configuración Inicial del Entorno Local

2.1. Paso 1: Instalación del Sistema Base

1. Windows con WSL2 (si aplica):

```

1 # En PowerShell como Administrador
2 wsl --install
3 wsl --set-default-version 2
4 # Instalar Ubuntu desde Microsoft Store
5 wsl -l -v # Verificar instalacion

```

2. Actualización del sistema (Linux/macOS):

```

1 # Ubuntu/Debian
2 sudo apt update && sudo apt upgrade -y
3 sudo apt install build-essential curl git -y
4
5 # macOS
6 xcode-select --install
7 brew update && brew upgrade

```

2.2. Paso 2: Instalación y Configuración de Python

```

1 # Verificar Python existente
2 python3 --version
3 pip3 --version
4
5 # Instalar Python 3.11 si es necesario (Ubuntu)
6 sudo apt install python3.11 python3.11-venv python3.11-dev -y
7
8 # Crear entorno virtual
9 python3.11 -m venv ~/adk-env
10 source ~/adk-env/bin/activate # Linux/macOS
11 # En Windows (PowerShell): ~\adk-env\Scripts\Activate.ps1
12
13 # Actualizar pip y herramientas basicas
14 pip install --upgrade pip setuptools wheel

```

2.3. Paso 3: Instalación de Google Cloud SDK

```

1 # Linux (Ubuntu)
2 echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] https://packages.cloud.
   google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud
   -sdk.list
3 curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key --keyring /
   usr/share/keyrings/cloud.google.gpg add -
4 sudo apt update && sudo apt install google-cloud-cli -y
5
6 # macOS
7 brew install --cask google-cloud-sdk
8
9 # Windows (PowerShell)

```

```

10 (New-Object Net.WebClient).DownloadFile("https://dl.google.com/dl/cloudsdk/channels/
    rapid/GoogleCloudSDKInstaller.exe", "$env:Temp\GoogleCloudSDKInstaller.exe")
11 & "$env:Temp\GoogleCloudSDKInstaller.exe"
12
13 # Inicializar y autenticar
14 gcloud init
15 gcloud auth application-default login
16 gcloud auth login

```

2.4. Paso 4: Configuración del Proyecto GCP

```

1 # Crear nuevo proyecto o usar existente
2 gcloud projects create mi-proyecto-adk --name="Proyecto ADK Local"
3 gcloud config set project mi-proyecto-adk
4
5 # Habilitar APIs necesarias
6 gcloud services enable aiplatform.googleapis.com
7 gcloud services enable cloudresourcemanager.googleapis.com
8 gcloud services enable iamcredentials.googleapis.com
9
10 # Configurar region por defecto
11 gcloud config set compute/region us-central1
12 gcloud config set compute/zone us-central1-a
13
14 # Crear cuenta de servicio
15 gcloud iam service-accounts create adk-local-sa \
    --display-name="Cuenta de servicio ADK Local"
16
17 # Otorgar permisos
18 gcloud projects add-iam-policy-binding mi-proyecto-adk \
    --member="serviceAccount:adk-local-sa@mi-proyecto-adk.iam.gserviceaccount.com" \
    --role="roles/aiplatform.user"
19
20 # Generar clave de servicio (JSON)
21 gcloud iam service-accounts keys create ~/adk-key.json \
    --iam-account=adk-local-sa@mi-proyecto-adk.iam.gserviceaccount.com
22
23 # Establecer variable de entorno
24 export GOOGLE_APPLICATION_CREDENTIALS=~/adk-key.json
25 # En Windows: $env:GOOGLE_APPLICATION_CREDENTIALS="C:\Users\Usuario\adk-key.json"

```

3. Instalación y Configuración del ADK

3.1. Paso 5: Instalación del Agent Development Kit

```

1 # Con entorno virtual activado
2 pip install google-cloud-aiagent
3 pip install google-cloud-aiplatform>=1.38
4
5 # Instalar dependencias adicionales recomendadas
6 pip install langchain
7 pip install langchain-google-vertexai
8 pip install google-generativeai
9 pip install pandas numpy matplotlib # Para procesamiento de datos
10 pip install jupyter notebook # Para experimentacion interactiva
11
12 # Verificar instalacion
13 python -c "import google.cloud.aiagent as aiagent; print(f'ADK Version: {aiagent.
    __version__}')"

```

3.2. Paso 6: Configuración del Entorno de Desarrollo

```
1 # Crear archivo de configuracion local: config_adk.py
2 import os
3 from google.oauth2 import service_account
4
5 # Configuracion base
6 PROJECT_ID = "mi-proyecto-adk"
7 LOCATION = "us-central1"
8 MODEL_NAME = "gemini-1.5-pro" # Alternativa: "gemini-1.5-flash"
9
10 # Ruta a las credenciales (ajustar segun tu sistema)
11 CREDENTIALS_PATH = os.path.expanduser("~/adk-key.json")
12
13 # Validar que el archivo existe
14 if not os.path.exists(CREDENTIALS_PATH):
15     raise FileNotFoundError(f"Credenciales no encontradas en: {CREDENTIALS_PATH}")
16
17 # Cargar credenciales
18 credentials = service_account.Credentials.from_service_account_file(
19     CREDENTIALS_PATH,
20     scopes=["https://www.googleapis.com/auth/cloud-platform"]
21 )
22
23 # Configurar variables de entorno
24 os.environ["GOOGLE_CLOUD_PROJECT"] = PROJECT_ID
25 os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = CREDENTIALS_PATH
26
27 print("Configuracion cargada correctamente")
28 print(f"Proyecto: {PROJECT_ID}")
29 print(f"Modelo: {MODEL_NAME}")
```

4. Desarrollo del Agente de IA

4.1. Paso 7: Crear un Agente Básico Local

```
1 # archivo: mi_agente_local.py
2 import os
3 import sys
4 from google.cloud import aiplatform
5 from google.cloud import aiagent
6 from google.cloud.aiagent import components
7
8 # Inicializar Vertex AI
9 aiplatform.init(
10     project=os.environ["GOOGLE_CLOUD_PROJECT"],
11     location="us-central1",
12     credentials=credentials # Del paso anterior
13 )
14
15 class MiAgenteLocal:
16     def __init__(self, model_name="gemini-1.5-pro"):
17         """Iniciar agente con componentes basicos"""
18
19         # 1. Crear instancia del agente
20         self.agent = aiagent.Agent(
21             name="agente-local-demo",
22             model=model_name,
23             instructions="""Eres un asistente especializado ejecutandose localmente.
24             Proporciona respuestas precisas y utiles.
25             Si no sabes algo, admítelo honestamente."""
26         )
```

```

28     # 2. Agregar memoria de conversacion
29     self.memory = components.ConversationMemory(
30         max_turns=20,
31         storage_path="../conversation_memory.json"
32     )
33     self.agent.add_component(self.memory)
34
35     # 3. Configurar planificador
36     self.planner = components.ReActPlanner()
37     self.agent.add_component(self.planner)
38
39     print(f"Agente inicializado con modelo: {model_name}")
40
41 def agregar_herramienta_busqueda(self, api_key=None, cse_id=None):
42     """Agregar herramienta de busqueda web (opcional)"""
43     if api_key and cse_id:
44         search_tool = components.WebSearchTool(
45             api_key=api_key,
46             custom_search_engine_id=cse_id
47         )
48         self.agent.add_tool(search_tool)
49         print("Herramienta de busqueda agregada")
50
51 def agregar_herramienta_codigo(self):
52     """Agregar herramienta para ejecutar codigo Python"""
53     code_tool = components.CodeInterpreterTool(
54         safe_execution=True,
55         timeout_seconds=30
56     )
57     self.agent.add_tool(code_tool)
58     print("Herramienta de codigo agregada")
59
60 def consultar(self, pregunta):
61     """Ejecutar consulta con el agente"""
62     print(f"\n[Usuario]: {pregunta}")
63     response = self.agent.run(pregunta)
64     print(f"\n[Agente]: {response.text}")
65     return response
66
67 def guardar_estado(self, ruta="../estado_agente.json"):
68     """Guardar estado del agente en disco"""
69     self.agent.save(ruta)
70     print(f"Estado guardado en: {ruta}")
71
72 def cargar_estado(self, ruta="../estado_agente.json"):
73     """Cargar estado del agente desde disco"""
74     if os.path.exists(ruta):
75         self.agent.load(ruta)
76         print(f"Estado cargado desde: {ruta}")
77
78 # Uso basico
79 if __name__ == "__main__":
80     # Crear instancia del agente
81     agente = MiAgenteLocal(model_name="gemini-1.5-flash")
82
83     # Ejemplo de consulta
84     respuesta = agente.consultar(
85         "Explica los beneficios de desarrollar agentes de IA localmente"
86     )
87
88     # Guardar historial
89     agente.guardar_estado()

```

4.2. Paso 8: Agente con Funciones Personalizadas

```

1 # archivo: agente_personalizado.py
2 from typing import Dict, Any
3 import json
4 from datetime import datetime
5
6 class AgentePersonalizado:
7     def __init__(self):
8         # Herramientas personalizadas
9         self.custom_tools = {
10             "calculadora": self._herramienta_calculadora,
11             "formateador_json": self._herramienta_json,
12             "obtener_fecha": self._herramienta_fecha
13         }
14
15     def _herramienta_calculadora(self, expresion: str) -> str:
16         """Evaluar expresion matematica segura"""
17         try:
18             # Solo operaciones matematicas basicas
19             allowed_chars = set("0123456789+-*/(). ")
20             if all(c in allowed_chars for c in expresion):
21                 result = eval(expresion)
22                 return f"Resultado: {result}"
23             else:
24                 return "Error: Expresion contiene caracteres no permitidos"
25         except Exception as e:
26             return f"Error en calculo: {str(e)}"
27
28     def _herramienta_json(self, data: str) -> str:
29         """Formatear y validar JSON"""
30         try:
31             parsed = json.loads(data)
32             return json.dumps(parsed, indent=2, ensure_ascii=False)
33         except json.JSONDecodeError as e:
34             return f"JSON invalido: {str(e)}"
35
36     def _herramienta_fecha(self, formato: str = "%Y-%m-%d") -> str:
37         """Obtener fecha actual"""
38         return datetime.now().strftime(formato)
39
40 # Integracion con ADK
41 def crear_agente_completo():
42     from google.cloud import aiagent
43
44     agent = aiagent.Agent(
45         name="agente-avanzado",
46         model="gemini-1.5-pro",
47         instructions="Usa las herramientas disponibles para ayudar al usuario"
48     )
49
50     # Agregar herramientas personalizadas via funciones
51     personalizado = AgentePersonalizado()
52
53     # Convertir herramientas personalizadas al formato ADK
54     for name, func in personalizado.custom_tools.items():
55         tool_wrapper = aiagent.Tool.from_function(
56             func=func,
57             name=name,
58             description=f"Herramienta para {name}"
59         )
60         agent.add_tool(tool_wrapper)
61
62     return agent

```

5. Pruebas y Validación

5.1. Paso 9: Suite de Pruebas Local

```
1 # archivo: test_agente.py
2 import unittest
3 from mi_agente_local import MiAgenteLocal
4
5 class TestAgenteLocal(unittest.TestCase):
6
7     @classmethod
8     def setUpClass(cls):
9         """Iniciar agente una vez para todas las pruebas"""
10        cls.agente = MiAgenteLocal(model_name="gemini-1.5-flash")
11
12    def test_inicializacion(self):
13        """Verificar que el agente se inicializa correctamente"""
14        self.assertIsNotNone(self.agente.agent)
15        self.assertIsNotNone(self.agente.memory)
16
17    def test_consulta_basica(self):
18        """Probar consulta simple"""
19        respuesta = self.agente.consultar("Hola, como estas?")
20        self.assertIsNotNone(respuesta)
21        self.assertIsInstance(respuesta.text, str)
22        self.assertGreater(len(respuesta.text), 10)
23
24    def test_memoria_conversacion(self):
25        """Verificar que la memoria funciona"""
26        pregunta1 = "Mi nombre es Carlos"
27        self.agente.consultar(pregunta1)
28
29        pregunta2 = "Cual es mi nombre?"
30        respuesta = self.agente.consultar(pregunta2)
31
32        # Verificar que recuerda el contexto
33        self.assertIn("Carlos", respuesta.text)
34
35    def test_guardado_carga(self):
36        """Probar persistencia del agente"""
37        ruta_test = "./test_estado.json"
38
39        # Guardar estado
40        self.agente.guardar_estado(ruta_test)
41
42        # Crear nuevo agente y cargar estado
43        nuevo_agente = MiAgenteLocal()
44        nuevo_agente.cargar_estado(ruta_test)
45
46        self.assertIsNotNone(nuevo_agente.agent)
47
48        # Limpiar archivo de prueba
49        import os
50        if os.path.exists(ruta_test):
51            os.remove(ruta_test)
52
53 if __name__ == "__main__":
54     unittest.main(verbosity=2)
```

5.2. Paso 10: Ejecución y Monitoreo

```
1 # Ejecutar el agente en modo interactivo
2 python -i mi_agente_local.py
3
```

```

4 # Ejecutar pruebas unitarias
5 python -m pytest test_agente.py -v
6
7 # Monitorear uso de recursos durante ejecucion
8 # Linux:
9 htop # En otra terminal
10 nvidia-smi -l 1 # Para GPU NVIDIA
11
12 # Windows:
13 # Abrir Administrador de Tareas -> Rendimiento
14
15 # Ejecutar con logging detallado
16 import logging
17 logging.basicConfig(level=logging.DEBUG)

```

6. Despliegue y Producción

6.1. Paso 11: Configuración para Producción

```

1 # archivo: deployment_config.py
2 import yaml
3
4 config_produccion = {
5     "version": "1.0",
6     "agent": {
7         "name": "agente-produccion",
8         "model": "gemini-1.5-pro",
9         "max_concurrent_requests": 10,
10        "timeout_seconds": 60,
11        "rate_limit": {
12            "requests_per_minute": 100,
13            "requests_per_day": 10000
14        }
15    },
16    "monitoring": {
17        "enable_logging": True,
18        "log_level": "INFO",
19        "metrics": ["latency", "success_rate", "token_usage"],
20        "alerting": {
21            "email": "admin@example.com",
22            "thresholds": {
23                "error_rate": 0.05,
24                "p99_latency": 5000 # ms
25            }
26        }
27    },
28    "security": {
29        "api_key_required": True,
30        "cors_origins": ["https://tudominio.com"],
31        "input_validation": True,
32        "output_sanitization": True
33    }
34}
35
36 # Guardar configuracion
37 with open("config_produccion.yaml", "w") as f:
38     yaml.dump(config_produccion, f, default_flow_style=False)

```

6.2. Paso 12: Contenerización (Opcional)

```

1 # Dockerfile
2 FROM python:3.11-slim

```

```

3
4 WORKDIR /app
5
6 # Instalar dependencias del sistema
7 RUN apt-get update && apt-get install -y \
8     curl \
9     git \
10    && rm -rf /var/lib/apt/lists/*
11
12 # Copiar requirements
13 COPY requirements.txt .
14 RUN pip install --no-cache-dir -r requirements.txt
15
16 # Copiar código de la aplicación
17 COPY . .
18
19 # Crear usuario no root
20 RUN useradd -m -u 1000 appuser && chown -R appuser:appuser /app
21 USER appuser
22
23 # Variables de entorno
24 ENV GOOGLE_APPLICATION_CREDENTIALS=/app/credentials/key.json
25 ENV PYTHONPATH=/app
26 ENV PORT=8080
27
28 # Exponer puerto
29 EXPOSE 8080
30
31 # Comando de inicio
32 CMD ["python", "app/main.py"]

```

7. Solución de Problemas Comunes

Problema	Causa Probable	Solución
Error de autenticación	Credenciales no configuradas o inválidas	Verificar GOOGLE_APPLICATION_CREDENTIALS, regenerar clave JSON
Falta de memoria	Modelo muy grande o muchas herramientas	Reducir tamaño de modelo, usar gemini-1.5-flash, aumentar swap
Lentitud en respuestas	Conexión lenta a APIs de Google	Usar modelo local (si GPU disponible), optimizar consultas
ImportError: módulo no encontrado	Entorno virtual no activado o dependencias faltantes	Activar entorno virtual, pip install -r requirements.txt
Límites de cuota excedidos	Uso excesivo de APIs de Google	Solicitar aumento de cuota, implementar caché, monitorear uso
Problemas con WSL2 en Windows	Configuración incorrecta de WSL2	wsl –update, aumentar memoria asignada en .wslconfig

Cuadro 2: Problemas comunes y soluciones en desarrollo local

8. Recursos y Referencias

- Documentación Oficial:

- Google ADK: <https://cloud.google.com/agent-development-kit>
- Vertex AI: <https://cloud.google.com/vertex-ai>
- Gemini API: <https://cloud.google.com/vertex-ai/generative-ai>

▪ **Repositorios de Ejemplo:**

- GitHub ADK: <https://github.com/google-cloud-ai/agent-development-kit>
- Ejemplos oficiales: <https://github.com/GoogleCloudPlatform/ai-agent-examples>

▪ **Comunidad y Soporte:**

- Stack Overflow: <https://stackoverflow.com/questions/tagged/google-cloud-ai>
- Google Cloud Community: <https://www.googlecloudcommunity.com>

▪ **Herramientas Adicionales:**

- LangChain: <https://python.langchain.com>
- LlamaIndex: <https://www.llamaindex.ai>
- Ollama (para modelos locales): <https://ollama.ai>

9. Conclusión

Desarrollar agentes de IA con Google ADK en una PC local ofrece mayor control, flexibilidad y privacidad. Esta guía proporciona los pasos necesarios desde la configuración del hardware hasta el despliegue de agentes funcionales. Para mantener el entorno actualizado, se recomienda:

1. Actualizar regularmente los paquetes Python: `pip list --outdated`
2. Revisar los cambios en las APIs de Google Cloud
3. Mantener backups de las configuraciones y credenciales
4. Monitorear el uso y costos en Google Cloud Console

Con esta configuración, podrás desarrollar, probar y refinar agentes de IA complejos directamente desde tu computadora personal antes de desplegarlos en entornos de producción.